Key-updatable public-key encryption with keyword search (Or: How to realize PEKS with efficient key updates for IoT environments)

doi: 10.1007/s10207-019-00441-2

Key-Updatable Public-Key Encryption with Keyword Search

Or: How to Realize PEKS with Efficient Key Updates for IoT Environments

Hiroaki Anada · Akira Kanaoka · Natsume Matsuzaki · Yohei Watanabe

Updated: April 25th, 2019

Abstract Security and privacy are the key issues for the Internet of Things (IoT) systems. Especially, secure search is an important functionality for cooperation among users' devices and non-trusted servers. *Public-key encryption with keyword search* (PEKS) enables us to search encrypted data, and is expected to be used between a cloud server and users' mobile devices or IoT devices. However, those mobile devices might be lost or stolen. For IoT devices, it might be difficult to store keys in a tamper-proof manner due to prohibitive costs. In this paper, we deal with such a key-exposure problem on PEKS, and introduce the concept of PEKS with key-updating functionality, which we call *key-updatable PEKS* (KU-PEKS). Specifically, we propose two models of KU-PEKS: the *key-evolution model* and the *key-insulation model*. In the key-evolution model, a pair of public and secret keys can be updated if needed (e.g., the secret key is exposed). In the key-insulation model, the public key remains fixed while the secret key can be updated if needed. The former model makes a construction simple and more efficient than the latter. On the other hand, the latter model is preferable for practical use since a user never updates their public key. We show constructions in each model in a black-box manner. We also give implementation results on Raspberry Pi 3, which can be regarded as a reasonable platform of IoT devices.

Keywords Searchable encryption · Public-key encryption with keyword search · Key updates · IoT environments · Raspberry Pi

The preliminary version of this paper was published in the proceedings of the 23rd Australasian Conference on Information Security and Privacy (ACISP 2018) [3]. This is the full version.

H. Anada · N. Matsuzaki
University of Nagasaki, Nagasaki, Japan
E-mail: {anada, matsuzaki}@sun.ac.jp

A. Kanaoka
Toho-University, Chiba, Japan
E-mail: akira.kanaoka@is.sci.toho-u.ac.jp

Y. Watanabe
The University of Electro-Communications, Tokyo, Japan
E-mail: watanabe@uec.ac.jp

Advanced Institute of National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan
Present address: of Y. Watanabe
National Institute of Information and Communications Technology (NICT), Tokyo, Japan
E-mail: yohei.watanabe@nict.go.jp

1 Introduction

Security and privacy are crucial for the Internet of Things (IoT) systems; especially, making search functionality secure is important for the cooperation among devices and non-trusted servers. *Public-key encryption with keyword search* (PEKS), proposed by Boneh et al. [7], enables a user to search encrypted data with keywords in a privacy-preserving way. PEKS is an efficient solution to the problem of constructing a private information retrieval (PIR) system [13,9]; for example, PEKS can be applied in a search system on an e-mail server. A user encrypts keywords related to each e-mail, such as "urgent", by PEKS as well as e-mails by S/MIME. Both e-mails and their corresponding keywords are stored in a database connected to the server. When the user wants to search for a keyword, they generate *a trapdoor* of the keyword. The server can check whether or not each stored e-mail contains the keyword while the server can get only negligible information on the e-mails and the keyword. Importantly, such a system might be implemented in mobile or IoT devices which could be lost or

stolen. Furthermore, the side-channel attack (e.g., [22]) is a class of powerful attacks that directly leak secret information such as secret keys. Therefore, we need a countermeasure against the leakage of those keys, however, it is unrealistic to make IoT devices tamper-proof as it involves significant costs.

In this paper, we are interested in resolving such a key-exposure problem on PEKS, and we consider key-updating functionality for PEKS. In fact, according to the NIST guideline SP800-57 [26], "re-keying", which we call "key update" in this paper, is one of the important factors affecting the length of a cryptoperiod.[1] Therefore, it is important to investigate this key-updating functionality for PEKS, however, to the best of our knowledge, little research has been done on it thus far. Abdalla et al. [2] considered public-key encryption with temporary keyword search (PETKS), in which the server can search ciphertexts encrypted during a cryptoperiod by using a trapdoor generated in the same cryptoperiod. Namely, the trapdoor is available only during the cryptoperiod, and therefore it reduces information leaked to the server. However, PETKS does not have key-updating functionality. Tang [29] proposed a PEKS scheme secure against the key exposure problem in the sense of forward security (not a PEKS scheme with certain key-updating functionality). The security relies on non-standard assumptions in composite-order groups, and therefore the resulting scheme is inefficient.

1.1 Our Contribution

In this paper, we introduce *key-updatable public-key encryption with keyword search* (KU-PEKS), which is the first PEKS with key-updating functionality.

The concept of KU-PEKS. First of all, we describe the concept of KU-PEKS, and clarify its requirements. Let us consider an e-mail service that automatically forwards an e-mail to user's smartphone if a keyword specified by the user, such as "urgent", is tagged to the e-mail. Suppose that a secret key generated by PEKS is stored in the smartphone, and then the user can specify the keyword by the trapdoor generated from the secret key and receive the corresponding e-mails without revealing any information on the keyword. The smartphone might be lost or stolen, and therefore the secret key (and its corresponding information) is updated periodically (or if needed). Then, taking into account practicality, the updated key should be used to search all encrypted keywords even if they were previously encrypted (i.e., encrypted in previous cryptoperiods). On the other hand, *old* trapdoors (i.e., trapdoors generated from the previous version of the secret key) should be useless to search for *new* keywords. Note that PETKS [2] and the Tang PEKS with forward security [29] do not have such functionality.

To wrap up, we consider the following three requirements of KU-PEKS.

Requirement 1: It is hard to guess an updated secret key from old (exposed) keys and public information (including the public key).

Requirement 2: Trapdoors generated from a new secret key can be used to search ciphertexts even if they were previously encrypted and old secret keys are already deleted. This requirement is for the purpose of availability.

Requirement 3: Trapdoors generated from a secret key are useless to search for keywords encrypted after the secret key is updated.

Our proposal. Based on the above requirements, we propose two models of KU-PEKS: a *key-evolution model* (Section 3) and a *key-insulation model* (Section 4). The former model is one of the most likely models of KU-PEKS, and the latter model is based on key-insulated cryptography introduced by Dodis et al. [15].

In the key-evolution model, a pair of public and secret keys can be updated if the secret key is exposed. This model makes a construction simple and efficient while not only the secret key but the public key have to be updated. Actually, we construct a KU-PEKS scheme in this model from any collision resistant hash function, any public-key encryption (PKE) scheme, and any PEKS scheme in a black-box manner.

In the key-insulation model, the public key remains fixed while the secret key can be updated if needed. Namely, this model is more practical than the key-evolution model in the sense of practical usage. To give a generic construction of a KU-PEKS scheme, we introduce a new key-insulated cryptographic protocol, a *key-insulated identity-based encryption for master keys* ("MIKE" for short), which has similar key-insulated functionality to key-insulated IBE [20,30]. MIKE realizes the *key-insulated functionality for master keys*. A master key for a cryptoperiod i generates the users' decryption key for i, which can be used to decrypt ciphertexts encrypted for i. Even if a master key for a cryptoperiod i is exposed, it does not affect master keys for other cryptoperiods (i.e., no information on master keys and decryption keys for other cryptoperiods is leaked from the exposed master key). We believe this new primitive is of independent interest. We show a generic construction of a KU-PEKS scheme in this

[1] A cryptoperiod [26] means the time span during which a specific key is authorized for use or in which the keys for a given system or application may remain in effect.

model from any collision-resistant hash function, any key-insulated PKE (KI-PKE) [15,28], and any anonymous MIKE scheme, which is instantiated under the symmetric external Diffie-Hellman (SXDH) assumption in Section 6.

Implementation using Raspberry Pi. To show practical efficiency, we implement our proposals on Raspberry Pi 3, which can be regarded as a reasonable platform for IoT devices, in Section 5. Specifically, we instantiate our generic constructions in the both models in three ways. The first instantiation (secure in the key-evolution model with random oracles) employs ElGamal PKE [17] and a PEKS scheme from Boneh-Franklin identity-based encryption (IBE) [8]. By employing existing anonymous IBE schemes secure in the standard model (e.g., [23]) instead of Boneh-Franklin IBE, we also obtain the second instantiation without random oracles. The third, which is secure in the key insulation model without random oracles, employs Watanabe-Shikata KI-PKE [28] and the proposed direct construction of an anonymous MIKE scheme (in Section 6).

The summary of our implementation results is as follows. The first (the key-evolution model) is the most efficient in terms of running time. On the other hand, the third (the key insulation model) requires relatively longer running time for the setup and key update algorithms, whereas, as mentioned earlier, the key insulation model is more practical in the sense that the public key is never changed. Nonetheless, we consider it applicable for IoT devices since those algorithms are rarely called.

1.2 Related Work

We should look into the three kinds of previous researches related our key-updating functionality: forward-secure, key-insulated, and intrusion-resilient cryptography. In forward-secure PKE, which was introduced by Canetti et al. [11], the secret key is updated for each cryptoperiod. Even if the secret key for a cryptoperiod i is exposed, ciphertexts encrypted before i still preserves the confidentiality. As described earlier, Tang [29] proposed forward-secure PEKS, however it is inefficient. In KI-PKE introduced by Dodis et al. [15], a receiver has two kinds of secret keys: a decryption and a helper key. These kinds of secret keys are stored in different devices such as a smartphone and USB pen drive. Unlike forward-secure PKE, the decryption key is updated with the aid of the helper key. The exposure of the decryption key for a cryptoperiod does not affect keys for other cryptoperiods; the exposed key is useless to get plaintexts encrypted during other cryptoperiods. Dodis et al. [14] developed intrusion-resilient PKE that simultaneously realizes forward security and key insulation. This rich security notion was attained paying the cost of computational amount and ciphertext length.

Proxy re-encryption (PRE) [5], especially identity-based PRE (IB-PRE) [19], might have a possibility of achieving a similar key-updating functionality. That is, KU-PEKS might be constructed from IB-PRE by regarding users in IB-PRE as cryptoperiods in KU-PEKS; re-encrypting a ciphertext for a user i into that for a user j can be considered as re-encrypting it for a cryptoperiod i into that for j. However, the underlying IB-PRE scheme must satisfy multi-hopping, unidirectionality, and anonymity to yield a KU-PEKS scheme satisfying our requirements. Unfortunately, there is no such an IB-PRE scheme so far, and this approach seems an inappropriate direction to pursue.

As for PEKS, researchers have studied additional functionality and security notions such as no use of secure channels [4] and the security against keyword guessing attacks [10]. Especially, Emura et al. gave the revocation functionality for trapdoors [18], but their scheme supports neither revocation nor key-updating functionality for *users' secret keys*.

1.3 Refinements

This paper is the full version of [3]. This version contains the following new results which did not appear in the proceedings version.

1. **Improvement of our constructions.** Previous constructions in the both models reveal the underlying keywords when re-encrypting ciphertexts. It does not violate the security definitions since an adversary (i.e., a malicious server) can get all the old secret keys, which means that it can decrypt all the old ciphertexts. However, it is problematic that even an honest server finds out the underlying keywords of old ciphertexts. In this version, to circumvent the problem, we improve the previous constructions by using collision-resistant hash function families. Specifically, keywords are hashed before encrypting them and generating trapdoors for them. In other words, secure search is performed via the encrypted hash value of keywords. Consequently, the server just finds out the hash values of the underlying keywords when re-encrypting them. Nonetheless, we note that this improvement does not perfectly resolve the problem since the server can still try to guess the underlying keywords by computing their hash values. In fact, we just prove

that the improved construction formally meets the same security definition as in [3].
2. **Implementation results for IoT environments.** In the proceedings version, we provided implementation for the first instantiation on normal environments (i.e., on PC). In this version, we implement all three instantiations on Raspberry Pi 3.
3. **A concrete anonymous MIKE scheme.** In the proceedings version, we skipped a direct construction of an anonymous MIKE scheme under the SXDH assumption. In this version, we give the full description of it as well as the security proof.

2 Preliminaries

Notation. For any positive integer i, let $[i]$ be $\{1, \ldots, i\}$. We denote by $x \leftarrow \mathcal{A}(y)$ assigning y to the input of a probabilistic polynomial time (PPT) algorithm $\mathcal{A}$ on an output x. We denote by $x \leftarrow \mathcal{A}^{\mathcal{O}}(y)$ $\mathcal{A}$ using an oracle $\mathcal{O}$ to output x. If S is a finite set, we denote by $x \xleftarrow{\$} S$ sampling x uniformly at random from S. Hereafter, we use the term of *time periods*, instead of cryptoperiods, as in the previous works [20,30], and let $\mathcal{T} := \{1, 2, \ldots, \mathrm{poly}(\lambda)\}$ be a set of time periods for simplicity.

2.1 Public-key Encryption

Public-key encryption (PKE) $\mathcal{PKE} = (\mathsf{PG}, \mathsf{G}, \mathsf{E}, \mathsf{D})$ is defined as follows.

- $\mathsf{PG}(1^\lambda) \to \mathsf{par}_{\mathrm{PKE}}$: given the security parameter 1^λ, it outputs a public parameter $\mathsf{par}_{\mathrm{PKE}}$.
- $\mathsf{G}(\mathsf{par}_{\mathrm{PKE}}) \to (\mathsf{ek}, \mathsf{dk})$: given $\mathsf{par}_{\mathrm{PKE}}$, it outputs an encryption key ek and a decryption key dk.
- $\mathsf{E}(\mathsf{ek}, m) \to \mathsf{ct}$: given ek and a plaintext $m \in \mathcal{M}$, it outputs a ciphertext ct, where $\mathcal{M}$ is a plaintext space determined by the security parameter.
- $\mathsf{D}(\mathsf{dk}, \mathsf{ct}) \to m$ or $\perp$: given dk and ct, it outputs m or $\perp$, where $\perp$ indicates decryption failure.

We require $\mathcal{PKE}$ satisfies the following correctness: For all $\lambda \in \mathbb{N}$, all $m \in \mathcal{M}$, all $\mathsf{par}_{\mathrm{PKE}} \leftarrow \mathsf{PG}(1^\lambda)$, and all $(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{G}(\mathsf{par}_{\mathrm{PKE}})$, it holds $\mathsf{D}(\mathsf{dk}, \mathsf{E}(\mathsf{ek}, m)) = m$.

We describe a security notion of indistinguishability against chosen plaintext attacks (IND-CPA). Let $\mathcal{A}$ be a PPT adversary, and we consider the following experiment $\mathsf{Exp}^{\mathsf{CPA}}_{\mathcal{PKE},\mathcal{A}}(1^\lambda)$.

$\mathsf{Exp}^{\mathsf{CPA}}_{\mathcal{PKE},\mathcal{A}}(1^\lambda)$
$\mathsf{par}_{\mathrm{PKE}} \leftarrow \mathsf{PG}(1^\lambda), \quad (\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{G}(\mathsf{par}_{\mathrm{PKE}})$
$(m_0^*, m_1^*, state) \leftarrow \mathcal{A}(\mathsf{par}_{\mathrm{PKE}}, \mathsf{ek})$ s.t. $|m_0^*| = |m_1^*|$
$b \xleftarrow{\$} \{0,1\}, \quad \mathsf{ct}_b^* \leftarrow \mathsf{E}(\mathsf{ek}, m_b^*)$
$b' \leftarrow \mathcal{A}(state, \mathsf{ct}_b^*)$
if $b' = b$ **return** 1 **else return** 0

Definition 1 (IND-CPA) *A PKE scheme $\mathcal{PKE}$ is said to be* IND-CPA *secure if for all PPT adversaries $\mathcal{A}$, its advantage* $\mathsf{Adv}^{\mathsf{CPA}}_{\mathcal{PKE},\mathcal{A}}(1^\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{CPA}}_{\mathcal{PKE},\mathcal{A}}(1^\lambda) = 1] - 1/2|$ *is negligible in λ.*

2.2 Key-Insulated Public-key Encryption

Key-insulated public-key encryption (KI-PKE) $\mathcal{KIE} = (\mathsf{KIKG}, \mathsf{KIUG}, \mathsf{KIU}, \mathsf{KIE}, \mathsf{KID})$ is defined as follows.

- $\mathsf{KIKG}(1^\lambda) \to (\mathsf{EK}, \mathsf{DK}_0, \mathsf{HK})$: given the security parameter 1^λ, it outputs an encryption key EK, an initial decryption key DK_0, and a helper key HK.
- $\mathsf{KIUG}(\mathsf{HK}, i) \to \mathsf{UP}_i$: given HK and a time period $i \in \mathcal{T}$, it outputs update information UP_i.
- $\mathsf{KIU}(\mathsf{DK}_{i'}, \mathsf{UP}_i) \to \mathsf{DK}_i$: given $\mathsf{DK}_{i'}$ for a time period $i' \in \mathcal{T}$ and UP_i, it outputs an updated decryption key DK_i for a time period $i \in \mathcal{T}$.
- $\mathsf{KIE}(\mathsf{EK}, m, i) \to \mathsf{C}_i$: given EK, a plaintext $m \in \mathcal{M}$, and a current time period $i \in \mathcal{T}$, it outputs a ciphertext C_i for i, where $\mathcal{M}$ is a plaintext space determined by λ.
- $\mathsf{KID}(\mathsf{DK}_i, \mathsf{C}_i) \to m$ or $\perp$: given DK_i for a time period $i \in \mathcal{T}$ and C_i for the same time period as input, it outputs m or $\perp$, where $\perp$ indicates decryption failure.

We require $\mathcal{KIE}$ satisfies the following correctness: For all $\lambda \in \mathbb{N}$, all $m \in \mathcal{M}$, all $(\mathsf{EK}, \mathsf{DK}_0, \mathsf{HK}) \leftarrow \mathsf{KIKG}(1^\lambda)$, and all $i \in \mathcal{T}$, it holds that $\mathsf{KID}(\mathsf{DK}_i, \mathsf{KIE}(\mathsf{EK}, m, i)) = m$, where DK_i is any decryption key for i correctly generated by KIUG and KIU.

We describe a KI-PKE version of IND-CPA, which is called IND-KI-CPA for short [15]. Let $\mathcal{A}$ be a PPT adversary, and we consider the following experiment $\mathsf{Exp}^{\mathsf{KI\text{-}CPA}}_{\mathcal{KIE},\mathcal{A}}(1^\lambda)$.

$\mathsf{Exp}^{\mathsf{KI\text{-}CPA}}_{\mathcal{KIE},\mathcal{A}}(1^\lambda)$
$(\mathsf{EK}, \mathsf{DK}_0, \mathsf{HK}) \leftarrow \mathsf{KIKG}(1^\lambda)$
$(m_0^*, m_1^*, i^*, state) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{EK})$ s.t. $|m_0^*| = |m_1^*|$
$b \xleftarrow{\$} \{0,1\}, \quad \mathsf{C}^*_{i^*,b} \leftarrow \mathsf{KIE}(\mathsf{EK}, m_b^*, i^*)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}}(state, \mathsf{C}^*_{i^*,b})$
if $b' = b$ **return** 1 **else return** 0

$\mathcal{A}$ can access an oracle $\mathcal{O}$, which is defined as follows:

Let $\mathcal{L} := \emptyset$. For a query $i \in \mathcal{T} \cup \{\star\}$, $\mathcal{O}$ returns DK_i by computing $\mathsf{KIU}(\mathsf{DK}_0, \mathsf{KIUG}(\mathsf{HK}, i))$ if $i \in \mathcal{T} \setminus \{i^*\}$ and $\star \notin \mathcal{L}$ and adds i to $\mathcal{L}$. Else if $i = \star$ and $\mathcal{L} = \emptyset$, it returns HK and adds $\star$ to $\mathcal{L}$. Otherwise, it returns $\perp$. This oracle captures that $\mathcal{A}$ can obtain either (a number of) decryption keys or the helper key (not both).

Definition 2 (IND-KI-CPA) *A KI-PKE scheme $\mathcal{KIE}$ is said to be* IND-KI-CPA *secure if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{KI\text{-}CPA}}_{\mathcal{KIE},\mathcal{A}}(1^\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{KI\text{-}CPA}}_{\mathcal{KIE},\mathcal{A}}(1^\lambda) = 1] - 1/2|$ is negligible in λ.*

2.3 Public-key Encryption with Keyword Search

Public-key encryption with keyword search (PEKS) $\mathcal{PEKS} = (\mathsf{Setup}_{\mathrm{PEKS}}, \mathsf{KeyGen}_{\mathrm{PEKS}}, \mathsf{Enc}_{\mathrm{PEKS}}, \mathsf{Trapdoor}_{\mathrm{PEKS}}, \mathsf{Test}_{\mathrm{PEKS}})$ is defined as follows.

- $\mathsf{Setup}_{\mathrm{PEKS}}(1^\lambda) \to \mathsf{par}_{\mathrm{PEKS}}$: given the security parameter 1^λ, it outputs a public parameter $\mathsf{par}_{\mathrm{PEKS}}$.
- $\mathsf{KeyGen}_{\mathrm{PEKS}}(\mathsf{par}_{\mathrm{PEKS}}) \to (\mathsf{mpk}, \mathsf{msk})$: given $\mathsf{par}_{\mathrm{PEKS}}$, it outputs a public key mpk and a secret key msk.
- $\mathsf{Enc}_{\mathrm{PEKS}}(\mathsf{mpk}, w) \to \mathsf{ct}_w$: given mpk and a keyword $w \in \mathcal{W}$, it outputs a ciphertext ct_w, where $\mathcal{W}$ is a keyword space determined by the security parameter.
- $\mathsf{Trapdoor}_{\mathrm{PEKS}}(\mathsf{mpk}, \mathsf{msk}, w') \to \mathsf{t}_{w'}$: given mpk, msk, and a keyword $w' \in \mathcal{W}$, it outputs a trapdoor $\mathsf{t}_{w'}$.
- $\mathsf{Test}_{\mathrm{PEKS}}(\mathsf{mpk}, \mathsf{t}_{w'}, \mathsf{ct}_w) \to 1$ or 0: given mpk, $\mathsf{t}_{w'}$, and ct_w, it outputs 1, which indicates "keyword match", or 0.

We require $\mathcal{PEKS}$ satisfies the following correctness: For all $\lambda \in \mathbb{N}$, all $w \in \mathcal{W}$, $\mathsf{par}_{\mathrm{PEKS}} \leftarrow \mathsf{Setup}_{\mathrm{PEKS}}(1^\lambda)$, all $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{KeyGen}_{\mathrm{PEKS}}(\mathsf{par}_{\mathrm{PEKS}})$, it holds $\mathsf{Test}_{\mathrm{PEKS}}(\mathsf{mpk}, \mathsf{Trapdoor}_{\mathrm{PEKS}}(\mathsf{mpk}, \mathsf{msk}, w), \mathsf{Enc}_{\mathrm{PEKS}}(\mathsf{mpk}, w)) \to 1$.

We describe security notions for $\mathcal{PEKS}$, indistinguishability against chosen keyword attacks (IND-CKA) and Computational Consistency.

For any PPT adversary $\mathcal{A}$, we consider the following experiment $\mathsf{Exp}^{\mathsf{CKA}}_{\mathcal{PEKS},\mathcal{A}}(1^\lambda)$ for IND-CKA security.

$\mathsf{Exp}^{\mathsf{CKA}}_{\mathcal{PEKS},\mathcal{A}}(1^\lambda)$

$\mathsf{par}_{\mathrm{PEKS}} \leftarrow \mathsf{Setup}_{\mathrm{PEKS}}(1^\lambda)$

$(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{KeyGen}_{\mathrm{PEKS}}(\mathsf{par}_{\mathrm{PEKS}})$

$(w_0^*, w_1^*, \mathit{state}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{TD}}}(\mathsf{par}_{\mathrm{PEKS}}, \mathsf{mpk})$ s.t. $|w_0^*| = |w_1^*|$

$b \xleftarrow{\$} \{0,1\}, \quad \mathsf{ct}^*_{w_b^*} \leftarrow \mathsf{Enc}_{\mathrm{PEKS}}(\mathsf{mpk}, w_b^*)$

$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{TD}}}(\mathit{state}, \mathsf{ct}^*_{w_b^*})$

if $b' = b$ **return** 1 **else return** 0

$\mathcal{A}$ can access an oracle $\mathcal{O}_{\mathrm{TD}}$ which receives $w (\in \mathcal{W} \setminus \{w_0^*, w_1^*\})$, and returns $\mathsf{Trapdoor}_{\mathrm{PEKS}}(\mathsf{msk}, w)$.

Definition 3 (IND-CKA [2]) *A PEKS scheme $\mathcal{PEKS}$ is said to be* IND-CKA *secure if for all PPT adversaries $\mathcal{A}$, its advantage $\mathsf{Adv}^{\mathsf{CKA}}_{\mathcal{PEKS},\mathcal{A}}(1^\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{CKA}}_{\mathcal{PEKS},\mathcal{A}}(1^\lambda) = 1] - 1/2|$ is negligible in λ.*

We next define Computational Consistency. We also consider the following experiment $\mathsf{Exp}^{\mathsf{Cons}}_{\mathcal{PEKS},\mathcal{A}}(1^\lambda)$ for any PPT adversary $\mathcal{A}$.

$\mathsf{Exp}^{\mathsf{Cons}}_{\mathcal{PEKS},\mathcal{A}}(1^\lambda)$

$\mathsf{par}_{\mathrm{PEKS}} \leftarrow \mathsf{Setup}_{\mathrm{PEKS}}(1^\lambda)$

$(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{KeyGen}_{\mathrm{PEKS}}(\mathsf{par}_{\mathrm{PEKS}})$

$(w_0^*, w_1^*) \leftarrow \mathcal{A}(\mathsf{par}_{\mathrm{PEKS}}, \mathsf{mpk})$ s.t. $|w_0^*| = |w_1^*|$

$\mathsf{ct}^*_{w_0^*} \leftarrow \mathsf{Enc}_{\mathrm{PEKS}}(\mathsf{mpk}, w_0^*)$

$\mathsf{t}_{w_1^*} \leftarrow \mathsf{Trapdoor}_{\mathrm{PEKS}}(\mathsf{msk}, w_1^*)$

if $\mathsf{Test}_{\mathrm{PEKS}}(\mathsf{t}_{w_1^*}, \mathsf{ct}_{w_0^*}) = 1$ and $w_0^* \neq w_1^*$ **return** 1

else return 0

Definition 4 (Computational Consistency [2]) *A PEKS scheme $\mathcal{PEKS}$ is said to be* IND-CKA *secure if for all PPT adversaries $\mathcal{A}$, its advantage $\mathsf{Adv}^{\mathsf{Cons}}_{\mathcal{PEKS},\mathcal{A}}(1^\lambda) := \Pr[\mathsf{Exp}^{\mathsf{Cons}}_{\mathcal{PEKS},\mathcal{A}}(1^\lambda) = 1]$ is negligible in λ.*

2.4 Hash Function Family

A hash function family $\mathcal{H}$ is a family of functions $\mathsf{H} : \mathcal{X} \to \mathcal{Y}$ that compress a string of an arbitrary length into a shorter constant string, and the size of $\mathcal{H}$ depends on a security parameter λ. We define Collision Resistance of $\mathcal{H}$ as follows.

$\mathsf{Exp}^{\mathsf{CR}}_{\mathcal{H},\mathcal{A}}(1^\lambda)$

$\mathsf{H} \xleftarrow{\$} \mathcal{H}$

$(x, x^*) \leftarrow \mathcal{A}(1^\lambda, \mathsf{H})$

if $(x, x^*) \in \mathcal{X}^2 \wedge x \neq x^* \wedge \mathsf{H}(x) = \mathsf{H}(x^*)$

then return 1 **else return** 0

Definition 5 (Collision Resistance) *A family of hash functions $\mathcal{H}$ is said to satisfy* Collision Resistance *if for all PPT adversaries $\mathcal{A}$, its advantage $\mathsf{Adv}^{\mathsf{CR}}_{\mathcal{H},\mathcal{A}}(1^\lambda) := \Pr[\mathsf{Exp}^{\mathsf{CR}}_{\mathcal{H},\mathcal{A}}(1^\lambda) = 1]$ is negligible in λ.*

3 KU-PEKS in the Key-evolution Model

We introduce the first framework of KU-PEKS, which is called a *key-evolution model*. Roughly speaking, in the key-evolution model, both of a public key and a secret key are updated periodically. We believe that this is one of the most likely models that ones naturally come up with "PEKS with key-updating functionality".

3.1 Model

KU-PEKS in the key-evolution model is executed as follows. A user first runs $\mathsf{KE.Setup}$ to generate a public key pk_1 and a secret key sk_1. An i-th key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ can be updated by $\mathsf{KE.Upd}$ if needed (i.e., sk_i is exposed), and $\mathsf{KE.Upd}$ outputs an updated key pair $(\mathsf{pk}_{i+1}, \mathsf{sk}_{i+1})$ and a re-encryption key $\mathsf{rk}_{i\to i+1}$. The re-encryption key $\mathsf{rk}_{i\to i+1}$ is sent to the server via a secure channel (we will explain how to use $\mathsf{rk}_{i\to i+1}$ later). Nobody (except for the user) can correctly guess the new secret key sk_{i+1} from old keys (**Requirement 1**). Suppose that the current time period is i. As in PEKS, another user who wants to store an encrypted keyword in a server executes $\mathsf{KE.Enc}$ with the i-th public key pk_i and a keyword w, and gets a ciphertext (or, an encrypted keyword) $\mathsf{c}^{(0)}_{w,i}$, which is stored in the server. To search a keyword w', the user runs $\mathsf{KE.Trapdoor}$ with sk_i and w' and gets a trapdoor $\mathsf{t}_{w',i}$, which is sent to the server via the secure channel. The server uses $\mathsf{t}_{w',i}$ to search for the keyword w' over stored ciphertexts. Specifically, $\mathsf{KE.Test}$ is executed with $\mathsf{t}_{w',i}$ and $\mathsf{c}^{(k)}_{w,j}$ such that $j+k=i$, where j indicates a time period when it is generated and k indicates the number of updates. $\mathsf{KE.Test}$ outputs 1 if $w'=w$ holds (i.e., the search keyword matches the encrypted keyword), or outputs 0 otherwise. Note that the server only gets correct search results if and only if $j+k=i$. In other words, $\mathsf{KE.Test}$ never outputs 1 if the trapdoor is old, i.e., $j+k>i$ (**Requirement 3**). The server returns the search result to the user. In addition, the server can use re-encryption keys to update ciphertexts encrypted during the previous time period. More specifically, the server updates a ciphertext $\mathsf{c}^{(k)}_{w,j}$ such that $j+k=i$, by running $\mathsf{KE.ReEnc}$ with $\mathsf{rk}_{i\to i+1}$, and gets an updated ciphertext $\mathsf{c}^{(k+1)}_{w,j}$. Therefore, the user can search previously-encrypted keywords (**Requirement 2**).

Formally, KU-PEKS in the key-evolution model Π_{KE} = ($\mathsf{KE.Setup}$, $\mathsf{KE.Upd}$, $\mathsf{KE.Enc}$, $\mathsf{KE.ReEnc}$, $\mathsf{KE.Trapdoor}$, $\mathsf{KE.Test}$) is defined as follows.

- $\mathsf{KE.Setup}(1^\lambda) \to (\mathsf{pk}_1, \mathsf{sk}_1)$: given the security parameter 1^λ, it outputs an initial key pair $(\mathsf{pk}_1, \mathsf{sk}_1)$.
- $\mathsf{KE.Upd}(\mathsf{pk}_i, \mathsf{sk}_i) \to (\mathsf{pk}_{i+1}, \mathsf{sk}_{i+1}, \mathsf{rk}_{i\to i+1})$: given a key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ for a time period $i \in \mathcal{T}$, it outputs an updated key pair $(\mathsf{pk}_{i+1}, \mathsf{sk}_{i+1})$ for a next time period $i+1 \in \mathcal{T}$ and a re-encryption key $\mathsf{rk}_{i\to i+1}$.
- $\mathsf{KE.Enc}(\mathsf{pk}_i, w) \to \mathsf{c}^{(0)}_{w,i}$: given the public key pk_i and a keyword $w \in \mathcal{W}$, it outputs a ciphertext $\mathsf{c}^{(0)}_{w,i}$. The superscript of the ciphertext indicates the number of updates, and hence is 0 at this point.
- $\mathsf{KE.ReEnc}(\mathsf{pk}_{i+1}, \mathsf{rk}_{i\to i+1}, \mathsf{c}^{(k)}_{w,j}) \to \mathsf{c}^{(k+1)}_{w,j}$ or $\perp$: given pk_{i+1}, the re-encryption key $\mathsf{rk}_{i\to i+1}$ and a ciphertext $\mathsf{c}^{(k)}_{w,j}$, it outputs an updated ciphertext $\mathsf{c}^{(k+1)}_{w,j}$ if $j+k=i$ holds.[2] Otherwise, it outputs $\perp$.
- $\mathsf{KE.Trapdoor}(\mathsf{pk}_i, \mathsf{sk}_i, w') \to \mathsf{t}_{w',i}$: given pk_i, the secret key sk_i, and a keyword $w' \in \mathcal{W}$, it outputs a trapdoor $\mathsf{t}_{w',i}$ (for time period $i \in \mathcal{T}$).
- $\mathsf{KE.Test}(\mathsf{pk}_i, \mathsf{t}_{w',i}, \mathsf{c}^{(k)}_{w,j}) \to 1$ or 0: given pk_i, a trapdoor $\mathsf{t}_{w',i}$, and a ciphertext $\mathsf{c}^{(k)}_{w,j}$, it outputs 1 if $w=w'$ and $j+k=i$. Otherwise, it outputs 0.

We require Π_{KE} satisfies the following correctness. For all $\lambda \in \mathbb{N}$, all $i \in \mathcal{T}$, all $j \in [i-1]$, all $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KE.Setup}(1^\lambda)$, all $(\mathsf{pk}_\ell, \mathsf{sk}_\ell, \mathsf{rk}_{\ell-1\to\ell}) \leftarrow \mathsf{KE.Upd}(\mathsf{pk}_{\ell-1}, \mathsf{sk}_{\ell-1})$ $(2 \le \ell \le i)$, and all $w \in \mathcal{W}$, it holds $\mathsf{KE.Test}(\mathsf{pk}_i, \mathsf{KE.Trapdoor}(\mathsf{pk}_i, \mathsf{sk}_i, w), \mathsf{c}^{(i-j)}_{w,j}) \to 1$, where $\mathsf{c}^{(i-j)}_{w,j} \leftarrow \mathsf{KE.ReEnc}(\mathsf{pk}_i, \mathsf{rk}_{i-1\to i}, \mathsf{KE.ReEnc}(\cdots \mathsf{KE.ReEnc}(\mathsf{pk}_{j+1}, \mathsf{rk}_{j\to j+1}, \mathsf{KE.Enc}(\mathsf{pk}_j, w))\cdots))$. It means that $\mathsf{KE.Test}$ always outputs 1 if the search keyword matches the encrypted keyword and the ciphertext is generated for j and updated $i-j$ times when the version of the secret key is i.

We next define security of KU-PEKS in the key-evolution model. We consider security against an honest-but-curious server that obtains all leaked secret keys and re-encryption keys. As in traditional PEKS, we consider notions of indistinguishability against chosen keyword attacks in the key-evolution model ($\mathsf{IND\text{-}KE\text{-}CKA}$) and computational consistency in the key-evolution model ($\mathsf{KE\text{-}Computational\ Consistency}$).

First, we define $\mathsf{IND\text{-}KE\text{-}CKA}$ security. Let $\mathcal{A}$ be a PPT adversary, and we consider the following $\mathsf{IND\text{-}KE\text{-}CKA}$ game.

$\mathsf{Exp}^{\mathsf{KE\text{-}CKA}}_{\Pi_{\mathrm{KE}},\mathcal{A}}(1^\lambda)$

$\mathsf{ctr} := 1, \quad (\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KE.Setup}(1^\lambda)$

$(w^*_0, w^*_1, state) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{KG}}, \mathcal{O}_{\mathrm{KL}}, \mathcal{O}_{\mathrm{TD}}}(\mathsf{pk}_1)$ s.t. $|w^*_0| = |w^*_1|$

$\mathsf{ctr}^* := \mathsf{ctr}, \ b \xleftarrow{\$} \{0,1\}, \ \mathsf{c}^{(0)}_{w^*_b, \mathsf{ctr}^*} \leftarrow \mathsf{KE.Enc}(\mathsf{pk}_{\mathsf{ctr}^*}, w^*_b)$

$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{KL}}, \mathcal{O}_{\mathrm{TD}}}(state, \mathsf{c}^{(0)}_{w^*_b, \mathsf{ctr}^*})$

if $b'=b$ **return** 1 **else return** 0

In this game, $\mathcal{A}$ can access a set of the following oracles $\mathcal{O}_{\mathrm{KG}}, \mathcal{O}_{\mathrm{KL}}, \mathcal{O}_{\mathrm{TD}}$.

$\mathcal{O}_{\mathrm{KG}}$: Initially, let $\mathcal{SK} := \emptyset$. For $\mathcal{A}$'s request, $\mathcal{O}_{\mathrm{KG}}$ computes $(\mathsf{pk}_{\mathsf{ctr}+1}, \mathsf{sk}_{\mathsf{ctr}+1}, \mathsf{rk}_{\mathsf{ctr}\to\mathsf{ctr}+1}) \leftarrow \mathsf{KE.Upd}(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$, and returns $(\mathsf{pk}_{\mathsf{ctr}+1}, \mathsf{rk}_{\mathsf{ctr}\to\mathsf{ctr}+1})$ to $\mathcal{A}$. Then, $\mathsf{sk}_{\mathsf{ctr}+1}$ is added to $\mathcal{SK}$, and $\mathsf{ctr} := \mathsf{ctr}+1$.

[2] For simplicity, we assume that the information of i, j, and k is attached to $\mathsf{t}_{w',i}$ and $\mathsf{c}^{(k)}_{w,j}$.

$\mathcal{O}_{\mathrm{KL}}$: For a query $i \in \mathcal{T}$, $\mathcal{O}_{\mathrm{KL}}$ returns $\mathsf{sk}_i \in \mathcal{SK}$ if $i < \mathsf{ctr}$. Otherwise, $\mathcal{O}_{\mathrm{KL}}$ returns $\perp$. Note that this oracle captures key leakage.

$\mathcal{O}_{\mathrm{TD}}$: For $(w, i) \in \mathcal{W} \times \mathcal{T}$, $\mathcal{O}_{\mathrm{TD}}$ returns $\mathsf{KE.Trapdoor}(\mathsf{sk}_i, w)$ if $i \leq \mathsf{ctr}$ and $(w, i) \notin \{(w_0^*, \mathsf{ctr}), (w_1^*, \mathsf{ctr})\}$. Otherwise, $\mathcal{O}_{\mathrm{TD}}$ returns $\perp$.

Definition 6 (IND-KE-CKA) *A KU-PEKS scheme Π_{KE} is said to be* IND-KE-CKA *secure if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{KE\text{-}CKA}}_{\Pi_{\mathrm{KE}},\mathcal{A}}(1^\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{KE\text{-}CKA}}_{\Pi_{\mathrm{KE}},\mathcal{A}}(1^\lambda) = 1] - 1/2|$ is negligible in λ.*

Next, we define KE-Computational Consistency. For any PPT adversary $\mathcal{A}$, we consider the following experiment.

$\mathsf{Exp}^{\mathsf{KE\text{-}Cons}}_{\Pi_{\mathrm{KE}},\mathcal{A}}(1^\lambda)$

$\mathsf{ctr} := 1, \quad (\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KE.Setup}(1^\lambda)$

$(w_0^*, w_1^*, j^*, k^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{KG}}, \mathcal{O}_{\mathrm{KL}}}(\mathsf{pk}_1)$

s.t. $|w_0^*| = |w_1^*|$ and $j^* + k^* = \mathsf{ctr}$

$\mathsf{ctr}^* := \mathsf{ctr}, \ \mathsf{c}^{(0)}_{w_0^*, j^*} \leftarrow \mathsf{KE.Enc}(\mathsf{pk}_{j^*}, w_0^*)$

for $\ell = 1$ **to** k^* **do**

$\quad \mathsf{c}^{(\ell)}_{w_0^*, j^*} \leftarrow \mathsf{KE.ReEnc}(\mathsf{pk}_{j^*+\ell}, \mathsf{rk}_{j^*+\ell-1 \to j^*+\ell}, \mathsf{c}^{(\ell-1)}_{w_0^*, j^*})$

$\mathsf{td}_{w_1^*, \mathsf{ctr}} \leftarrow \mathsf{KE.Trapdoor}(\mathsf{pk}_{\mathsf{ctr}^*}, \mathsf{sk}_{\mathsf{ctr}^*}, w_1^*)$

if $\mathsf{KE.Test}(\mathsf{td}_{w_1^*, \mathsf{ctr}^*}, \mathsf{c}^{(k^*)}_{w_0^*, j^*}) = 1 \wedge w_0^* \neq w_1^*$ **return** 1

else return 0

In the above game, $\mathcal{A}$ can access $\mathcal{O}_{\mathrm{KG}}$ and $\mathcal{O}_{\mathrm{KL}}$, which are the same as the IND-KE-CKA game.

Definition 7 (KE-Computational Consistency) *A KU-PEKS scheme Π_{KE} is said to meet KE-Computational Consistency if for all PPT adversaries $\mathcal{A}$, its advantage $\mathsf{Adv}^{\mathsf{KE\text{-}Cons}}_{\Pi_{\mathrm{KE}},\mathcal{A}}(1^\lambda) := \Pr[\mathsf{Exp}^{\mathsf{KE\text{-}Cons}}_{\Pi_{\mathrm{KE}},\mathcal{A}}(1^\lambda) = 1]$ is negligible in λ.*

3.2 Generic Construction

In this section, we show a generic construction of a KU-PEKS scheme Π_{KE} in the key-evolution model from a PKE scheme, a PEKS scheme, and a family of hash functions. Let $\mathcal{PKE} = (\mathsf{PG}, \mathsf{G}, \mathsf{E}, \mathsf{D})$, $\mathcal{PEKS} = (\mathsf{Setup}_{\mathrm{PEKS}}, \mathsf{KeyGen}_{\mathrm{PEKS}}, \mathsf{Enc}_{\mathrm{PEKS}}, \mathsf{Trapdoor}_{\mathrm{PEKS}}, \mathsf{Test}_{\mathrm{PEKS}})$ and $\mathcal{H}$ be a PKE scheme, a PEKS scheme, and a hash function family, respectively. Our construction is given in Fig. 1.

Theorem 1 *If $\mathcal{PKE}$ is* IND-CPA *secure, $\mathcal{H}$ meets* Collision Resistance*, and $\mathcal{PEKS}$ is* IND-CKA *secure and meets* Computational Consistency*, the KU-PEKS scheme Π_{KE} given in Fig. 1 is* IND-KE-CKA *secure and meets* KE-Computational Consistency*.*

Proof Let G_0 be $\mathsf{Exp}^{\mathsf{KE\text{-}CKA}}_{\Pi_{\mathrm{KE}},\mathcal{A}}$, and we write $\mathsf{Exp}^{\mathsf{KE\text{-}CKA}}_{\Pi_{\mathrm{KE}},\mathcal{A}} \to 1$ (i.e., an event that $b' = b$ in G_0) as S_0. We define the following games $\mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_3$, and also define $\mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3$ in the same manner.

Let Col be an event that in the challenge phase, $\mathcal{A}$ submits w_0^* and w_1^* such that $\mathsf{H}(w_0^*) = \mathsf{H}(w_1^*)$. Then, we can easily show that $\Pr[\mathsf{S}_0 \wedge \mathsf{Col}] = \mathsf{Adv}^{\mathsf{CR}}_{\mathcal{H},\mathcal{B}}(1^\lambda)$.

G_1: This is the same as G_0 except that Col does not occur and the challenger guesses $i^* \in \mathcal{T}$ such that $\mathsf{ctr}^* = i^*$ at the beginning of the game. This modification does not affect any distributions, and therefore $\Pr[\mathsf{S}_0 \wedge \neg\mathsf{Col}] = \Pr[\mathsf{S}_1]$. Let Fail be an event that $\mathsf{ctr}^* \neq i^*$ (i.e., an event that the challenger eventually fails to guess correct i^*). Note that Fail and S_1 are independent.

G_2: This is the same as G_1 except that the challenger replaces a random bit with $\mathcal{A}$'s output b' regardless of $\mathcal{A}$'s actual output if Fail occurs. Since $\Pr[\mathsf{S}_1] = \Pr[\mathsf{S}_1 \mid \neg\mathsf{Fail}] = \Pr[\mathsf{S}_2 \mid \neg\mathsf{Fail}]$, we have $\Pr[\mathsf{S}_1] - 1/2 = \Pr[\mathsf{S}_2 \mid \neg\mathsf{Fail}] - 1/2$.

G_3: This is the same as G_2 except that in the challenge phase, the challenger computes $\mathsf{ct}^*_{\mathsf{ctr}} \leftarrow \mathsf{E}(\mathsf{ek}_{\mathsf{ctr}}, 0^{|w_b^*|})$ instead of $\mathsf{ct}^*_{\mathsf{ctr}} \leftarrow \mathsf{E}(\mathsf{ek}_{\mathsf{ctr}}, w_b^*)$. Then, we have

$$\Pr[\mathsf{S}_2 \mid \neg\mathsf{Fail}] - \frac{1}{2} = \Pr[\mathsf{S}_2 \mid \neg\mathsf{Fail}] - \Pr[\mathsf{S}_3 \mid \neg\mathsf{Fail}] + \Pr[\mathsf{S}_3 \mid \neg\mathsf{Fail}] - \frac{1}{2}.$$

Then, the rest of the proof follows the following lemmas. The proofs are given in Appendices A and B, respectively.

Lemma 1 $|\Pr[\mathsf{S}_2 \mid \neg\mathsf{Fail}] - \Pr[\mathsf{S}_3 \mid \neg\mathsf{Fail}]| = 2 \cdot \mathsf{Adv}^{\mathsf{CPA}}_{\mathcal{PKE},\mathcal{B}}(1^\lambda)$.

Lemma 2 $|\Pr[\mathsf{S}_3 \mid \neg\mathsf{Fail}] - 1/2| = \mathsf{Adv}^{\mathsf{CKA}}_{\mathcal{PEKS},\mathcal{B}}(1^\lambda)$.

Hence, we have

$$\mathsf{Adv}^{\mathsf{KE\text{-}CKA}}_{\Pi_{\mathrm{KE}},\mathcal{A}}(1^\lambda) \leq \mathsf{Adv}^{\mathsf{CR}}_{\mathcal{H},\mathcal{B}}(1^\lambda) + 2 \cdot \mathsf{Adv}^{\mathsf{CPA}}_{\mathcal{PKE},\mathcal{B}}(1^\lambda) + \mathsf{Adv}^{\mathsf{CKA}}_{\mathcal{PEKS},\mathcal{B}}(1^\lambda).$$

Similarly, we can show KE-Computational Consistency. Let G'_0 be $\mathsf{Exp}^{\mathsf{KE\text{-}Cons}}_{\Pi_{\mathrm{KE}},\mathcal{A}}$, and we write $\mathsf{Exp}^{\mathsf{KE\text{-}Cons}}_{\Pi_{\mathrm{KE}},\mathcal{A}} \to 1$ as S'_0. We define the following game G'_1 and also define S'_1 in the same manner.

Let Col be the same event as above. Namely, in the challenge phase, $\mathcal{A}$ submits w_0^* and w_1^* such that $\mathsf{H}(w_0^*) = \mathsf{H}(w_1^*)$. We then have $\Pr[\mathsf{S}'_0 \wedge \mathsf{Col}] = \mathsf{Adv}^{\mathsf{CR}}_{\mathcal{H},\mathcal{B}}(1^\lambda)$.

G'_1: This is the same as G'_0 except that Col does not occur and the challenger guesses $i^* \in \mathcal{T}$ such that $\mathsf{ctr}^* = i^*$ at the beginning of the game. This modification does not affect any distributions, and therefore $\Pr[\mathsf{S}'_0 \wedge \neg\mathsf{Col}] = \Pr[\mathsf{S}'_1]$. Let Fail be the same event as above. Note that Fail and S'_1 are independent, and

$\underline{\mathsf{KE.Setup}(1^\lambda)}$:
$\mathsf{par}_{\mathrm{PKE}} \leftarrow \mathsf{PG}(1^\lambda)$
$\mathsf{par}_{\mathrm{PEKS}} \leftarrow \mathsf{Setup}_{\mathrm{PEKS}}(1^\lambda)$
$\mathsf{H} \xleftarrow{\$} \mathcal{H}$
$(\mathsf{ek}_1, \mathsf{dk}_1) \leftarrow \mathsf{G}(\mathsf{par}_{\mathrm{PKE}})$
$(\mathsf{mpk}_1, \mathsf{msk}_1) \leftarrow \mathsf{KeyGen}_{\mathrm{PEKS}}(\mathsf{par}_{\mathrm{PEKS}})$
$\mathsf{pk}_1 := (\mathsf{par}_{\mathrm{PKE}}, \mathsf{par}_{\mathrm{PEKS}}, \mathsf{H}, \mathsf{ek}_1, \mathsf{mpk}_1)$
$\mathsf{sk}_1 := (\mathsf{dk}_1, \mathsf{msk}_1)$
return $(\mathsf{pk}_1, \mathsf{sk}_1)$

$\underline{\mathsf{KE.Upd}(\mathsf{pk}_i, \mathsf{sk}_i)}$:
parse $\mathsf{pk}_i = (\mathsf{par}_{\mathrm{PKE}}, \mathsf{par}_{\mathrm{PEKS}}, \mathsf{H}, \mathsf{ek}_i, \mathsf{mpk}_i)$
parse $\mathsf{sk}_i = (\mathsf{dk}_i, \mathsf{msk}_i)$
$(\mathsf{ek}_{i+1}, \mathsf{dk}_{i+1}) \leftarrow \mathsf{G}(\mathsf{par}_{\mathrm{PKE}})$
$(\mathsf{mpk}_{i+1}, \mathsf{msk}_{i+1}) \leftarrow \mathsf{KeyGen}_{\mathrm{PEKS}}(\mathsf{par}_{\mathrm{PEKS}})$
$\mathsf{pk}_{i+1} := (\mathsf{par}_{\mathrm{PKE}}, \mathsf{par}_{\mathrm{PEKS}}, \mathsf{H}, \mathsf{ek}_{i+1}, \mathsf{mpk}_{i+1})$
$\mathsf{sk}_{i+1} := (\mathsf{dk}_{i+1}, \mathsf{msk}_{i+1})$
$\mathsf{rk}_{i \to i+1} := \mathsf{dk}_i$
return $(\mathsf{pk}_{i+1}, \mathsf{sk}_{i+1}, \mathsf{rk}_{i \to i+1})$

$\underline{\mathsf{KE.Enc}(\mathsf{pk}_i, w)}$:
parse $\mathsf{pk}_i = (\mathsf{par}_{\mathrm{PKE}}, \mathsf{par}_{\mathrm{PEKS}}, \mathsf{H}, \mathsf{ek}_i, \mathsf{mpk}_i)$
$\mathsf{ct}_i \leftarrow \mathsf{E}(\mathsf{ek}_i, \mathsf{H}(w))$
// $\mathcal{M}$ (of $\mathcal{PKE}$) := $\mathcal{Y}$ (of $\mathcal{H}$)
$\mathsf{ct}_{w,i} \leftarrow \mathsf{Enc}_{\mathrm{PEKS}}(\mathsf{mpk}_i, \mathsf{H}(w))$
$\mathsf{c}^{(0)}_{w,i} := (\mathsf{ct}_i, \mathsf{ct}_{w,i})$
return $\mathsf{c}^{(0)}_{w,i}$

$\underline{\mathsf{KE.ReEnc}(\mathsf{pk}_{i+1}, \mathsf{rk}_{i \to i+1}, \mathsf{c}^{(k)}_{w,j})}$:
parse $\mathsf{rk}_{i \to i+1} = (\mathsf{ek}_{i+1}, \mathsf{mpk}_{i+1}, \mathsf{dk}_i)$
parse $\mathsf{c}^{(k)}_{w,j} = (\mathsf{ct}_{j+k}, \mathsf{ct}_{w,j+k})$
if $i \neq j + k$
  return $\perp$
else
  $\mathsf{H}(w) \leftarrow \mathsf{D}(\mathsf{dk}_i, \mathsf{ct}_i)$
  $\mathsf{ct}_{i+1} \leftarrow \mathsf{E}(\mathsf{ek}_{i+1}, \mathsf{H}(w))$
  $\mathsf{ct}_{w,i+1} \leftarrow \mathsf{Enc}_{\mathrm{PEKS}}(\mathsf{mpk}_{i+1}, \mathsf{H}(w))$
  $\mathsf{c}^{(k+1)}_{w,j} := (\mathsf{ct}_{i+1}, \mathsf{ct}_{w,i+1})$
  return $\mathsf{c}^{(k+1)}_{w,j}$

$\underline{\mathsf{KE.Trapdoor}(\mathsf{pk}_i, \mathsf{sk}_i, w')}$:
parse $\mathsf{sk}_i = (\mathsf{dk}_i, \mathsf{msk}_i)$
$\mathsf{t}_{w',i} \leftarrow \mathsf{Trapdoor}_{\mathrm{PEKS}}(\mathsf{mpk}_i, \mathsf{msk}_i, \mathsf{H}(w'))$
return $\mathsf{t}_{w',i}$

$\underline{\mathsf{KE.Test}(\mathsf{pk}_i, \mathsf{t}_{w',i}, \mathsf{c}^{(k)}_{w,j})}$:
parse $\mathsf{c}^{(k)}_{w,j} = (\mathsf{ct}_{j+k}, \mathsf{ct}_{w,j+k})$
if $i \neq j + k$
  return 0
else if $1 \leftarrow \mathsf{Test}_{\mathrm{PEKS}}(\mathsf{mpk}_i, \mathsf{t}_{w',i}, \mathsf{ct}_{w,i})$
  return 1
else if $0 \leftarrow \mathsf{Test}_{\mathrm{PEKS}}(\mathsf{mpk}_i, \mathsf{t}_{w',i}, \mathsf{ct}_{w,i})$
  return 0

Fig. 1 A generic construction of Π_{KE} from $\mathcal{PKE}$, $\mathcal{PEKS}$, and $\mathcal{H}$.

therefore we have $\Pr[\mathsf{S}'_1] = \Pr[\mathsf{S}'_1 \mid \mathsf{Fail}] = \Pr[\mathsf{S}'_1 \mid \neg\mathsf{Fail}]$. It is easy to see that $\Pr[\mathsf{S}'_1 \mid \neg\mathsf{Fail}] = \mathsf{Adv}^{\mathsf{Cons}}_{\mathcal{B},\mathcal{PEKS}}$, and we have

$$\mathsf{Adv}^{\mathsf{KE\text{-}Cons}}_{\Pi_{\mathrm{KE}},\mathcal{A}}(1^\lambda) = \mathsf{Adv}^{\mathsf{CR}}_{\mathcal{H},\mathcal{B}}(1^\lambda) + \mathsf{Adv}^{\mathsf{Cons}}_{\mathcal{PEKS},\mathcal{B}}(1^\lambda). \qquad \square$$

Remark 1 As noted in the introduction, the difference between the above construction and the previous one [3] is whether the hash function is used or not. Thanks to the underlying hash function, keywords are not directly revealed in KE.ReEnc in Fig. 1. However, the server can still guess the keywords with additional computational cost (i.e., computing their hash values to check if they match or not). Therefore, we can somehow argue that the above construction is better than the previous one, however, formally it just achieves the same level of security as in [3]. The same holds for our second construction in the next section.

4 KU-PEKS in the Key-insulation Model

Taking into account practical usage, it is desirable to keep the same public key for the updated secret key. In this section, we adopt the concept of *key-insulated cryptography* [15,16], which is an well-known cryptographic solution to the key-exposure problem, and propose the *key-insulation model* as another model of KU-PEKS. The key-insulation model achieves the property that a public key remains the same while the secret key is updated.

4.1 Model

The key-insulated protocol is said to have *random access key updates* [12] if one can update any old secret key to the latest version, more generally, if one can update a secret key from any time period $j \in \mathcal{T}$ to any time period $i \in \mathcal{T}$. Since the functionality of random access key updates is a basic requirement in key-insulated cryptography, we also consider it in this paper. Therefore, random access key updates eliminate the need for sequentially updating keys (i.e., $\mathsf{sk}_{i-1} \to \mathsf{sk}_i$), and thus allow the server to manage only one "global" time-period set $\mathcal{T}$ among all users (e.g., $t_1 := 1/10/2018, t_2 := 2/10/2018, \ldots$). Note that in the key-evolution model, the server has to manage different time-period sets per each user (i.e., a time period set is a counter of updates for each user). We also model re-encryption keys so that it updates ciphertexts from any time period to any time period since secret keys are not sequentially updated.

KU-PEKS in the key-insulation model is executed as follows. A user first runs KI.Setup to generate a public key pk, an initial secret key sk_0, and a helper key hk. sk_0 is stored in a (relatively) powerful but insecure device such as smartphones, and hk is stored in a physically-secure but computationally-limited device such as USB pen drives. The secret key $\mathsf{sk}_{i'}$ for a time period $i' \in \mathcal{T}$ is updated for any time period $i \in \mathcal{T}$ by Δ-Gen and KI.Upd if needed. Specifically, the user run Δ-Gen to compute update information δ_i for i with hk stored in the physically-secure devise. They then executes KI.Upd with δ_i to update $\mathsf{sk}_{i'}$ to sk_i. The above procedure can be done offline. KI.Upd also outputs a re-encryption key

rk_i, which is sent to the server via a secure channel. Due to the security definition, an adversary $\mathcal{A}$ is only allowed to get either the helper key hk or (a number of) decryption keys $\{\mathsf{sk}_{i_1}, \mathsf{sk}_{i_2}, \ldots, \mathsf{sk}_{i_q}\}$. Therefore, $\mathcal{A}$ cannot correctly update exposed keys since $\mathcal{A}$ can only execute either Δ-Gen or KI.Upd. It means that **Requirement 1** is satisfied. The flows of encryption, trapdoor generation, test, and re-encryption are almost the same as the key-evolution model (Note that any old ciphertext $\mathsf{ct}_{w,j}$ $(j < i)$ can be updated by rk_i in this model). Therefore, the key-insulation model also satisfies **Requirements 2** and **3**.

We formally define KU-PEKS in the key-insulation model $\Pi_{\mathrm{KI}} = (\mathsf{KI.Setup}, \Delta\text{-Gen}, \mathsf{KI.Upd}, \mathsf{KI.Enc}, \mathsf{KI.ReEnc}, \mathsf{KI.Trapdoor}, \mathsf{KI.Test})$.

- $\mathsf{KI.Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk}_0, \mathsf{hk})$: given the security parameter 1^λ, it outputs a public key pk, an initial secret key sk_0, and a helper key hk.
- $\Delta\text{-Gen}(\mathsf{pk}, \mathsf{hk}, i) \to \delta_i$: given pk, hk, and a time period $i \in \mathcal{T}$, it outputs update information δ_i for i.
- $\mathsf{KI.Upd}(\mathsf{pk}, \mathsf{sk}_{i'}, \delta_i) \to (\mathsf{sk}_i, \mathsf{rk}_i)$: given pk, a secret key $\mathsf{sk}_{i'}$ for a time period $i' \in \mathcal{T}$ and δ_i for $i \in \mathcal{T}$, it outputs an updated secret key sk_i and a re-encryption key rk_i.
- $\mathsf{KI.Enc}(\mathsf{pk}, w, i) \to \mathsf{c}_{w,i}$: given pk, a keyword $w \in \mathcal{W}$, and a current time period $i \in \mathcal{T}$, it outputs a ciphertext $\mathsf{c}_{w,i}$.
- $\mathsf{KI.ReEnc}(\mathsf{pk}, \mathsf{rk}_i, \mathsf{c}_{w,j}) \to \mathsf{c}_{w,i}$ or $\perp$: given pk, the re-encryption key rk_i for $i \in \mathcal{T}$, and a ciphertext $\mathsf{c}_{w,j}$ encrypted during $j \in \mathcal{T}$, it outputs an updated ciphertext $\mathsf{c}_{w,i}$ for i.
- $\mathsf{KI.Trapdoor}(\mathsf{pk}, \mathsf{sk}_i, w') \to \mathsf{td}_{w',i}$: given pk, a secret key sk_i for $i \in \mathcal{T}$, and a keyword $w' \in \mathcal{W}$, it outputs a trapdoor $\mathsf{td}_{w',i}$ for i.
- $\mathsf{KI.Test}(\mathsf{pk}, \mathsf{td}_{w',i}, \mathsf{c}_{w,i}) \to 1$ or 0: given pk, a trapdoor $\mathsf{td}_{w',i}$, and a ciphertext $\mathsf{c}_{w,i}$, it outputs 1 if $w = w'$. Otherwise, it outputs 0.

We require Π_{KI} satisfies the following correctness. For all $\lambda \in \mathbb{N}$, all $i, j \in \mathcal{T}$, all $(\mathsf{pk}, \mathsf{sk}_0, \mathsf{hk}) \leftarrow \mathsf{KI.Setup}(1^\lambda)$, and all $w \in \mathcal{W}$, it holds $\mathsf{KI.Test}(\mathsf{pk}, \mathsf{KI.Trapdoor}(\mathsf{pk}, \mathsf{sk}_i, w), \mathsf{c}_{w,i}) \to 1$, where sk_i is any secret key correctly updated from sk_0, and $\mathsf{c}_{w,i}$ is: (i) if $j = i$, $\mathsf{c}_{w,i} \leftarrow \mathsf{KI.Enc}(\mathsf{pk}, w, i)$; (ii) if $j \neq i$, $\mathsf{c}_{w,i} \leftarrow \mathsf{KI.ReEnc}(\mathsf{pk}, \mathsf{rk}_i, \mathsf{KI.ReEnc}(\cdots \mathsf{KI.Enc}(\mathsf{pk}, w, j) \cdots))$. It means that KI.Test always outputs 1 if the search keyword matches the encrypted keyword and the ciphertext is (correctly updated to) the same version of the secret key.

We next define security of KU-PEKS in the key-insulation model. As in the key-evolution model, we consider security against an honest-but-curious server that obtains all leaked secret keys and re-encryption keys, that is, we define notions of indistinguishability against chosen keyword attacks in the key-insulation model (IND-KI-CKA) and computational consistency in the key-insulation model (KI-Computational Consistency).

First, we define IND-KI-CKA security. Let $\mathcal{A}$ be a PPT adversary, and we consider the following experiment.

$\mathsf{Exp}^{\mathsf{KI\text{-}CKA}}_{\Pi_{\mathrm{KI}},\mathcal{A}}(1^\lambda)$
$(\mathsf{pk}, \mathsf{sk}_0, \mathsf{hk}) \leftarrow \mathsf{KI.Setup}(1^\lambda)$
$(w_0^*, w_1^*, i^*, \mathit{state}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{KL}}, \mathcal{O}_{\mathrm{RK}}, \mathcal{O}_{\mathrm{TD}}}(\mathsf{pk})$ s.t. $|w_0^*| = |w_1^*|$
$b \xleftarrow{\$} \{0,1\}$, $\mathsf{c}_{w_b^*, i^*} \leftarrow \mathsf{KI.Enc}(\mathsf{pk}, w_b^*, i^*)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{KL}}, \mathcal{O}_{\mathrm{RK}}, \mathcal{O}_{\mathrm{TD}}}(\mathit{state}, \mathsf{c}_{w_b^*, i^*})$
if $b' = b$ **return** 1 **else return** 0

$\mathcal{A}$ can access sets of oracles $\mathcal{O}_{\mathrm{KL}}, \mathcal{O}_{\mathrm{RK}}, \mathcal{O}_{\mathrm{TD}}$, and each oracle is defined as follows. Initially, it sets $\mathcal{L} := \emptyset$ and $\mathcal{RK} := \emptyset$.

$\mathcal{O}_{\mathrm{KL}}$: For a query $i \in \mathcal{T} \cup \{\star\}$, if $i \notin \mathcal{T} \setminus \{i^*\}$ and $\star \notin \mathcal{L}$, $\mathcal{O}_{\mathrm{KL}}$ computes $(\mathsf{sk}_i, \mathsf{rk}_i) \leftarrow \mathsf{KI.Upd}(\mathsf{pk}, \mathsf{sk}_0, \Delta\text{-Gen}(\mathsf{pk}, \mathsf{hk}, i))$, returns sk_i, and adds i and rk_i to $\mathcal{L}$ and $\mathcal{RK}$, respectively. Else if $i = \star$ and $\mathcal{L} = \emptyset$, $\mathcal{O}_{\mathrm{KL}}$ then returns hk and adds $\star$ to $\mathcal{L}$. Otherwise, $\mathcal{O}_{\mathrm{KL}}$ returns $\perp$. Note that this oracle captures key leakage, and $\mathcal{A}$ obtains either (a number of) decryption keys or the helper key during the game.

$\mathcal{O}_{\mathrm{RK}}$: For a query $i \in \mathcal{T}$, $\mathcal{O}_{\mathrm{RK}}$ returns $\mathsf{rk}_i \in \mathcal{RK}$ if $i \in \mathcal{L}$.[3]

$\mathcal{O}_{\mathrm{TD}}$: For $(w, i) \in \mathcal{W} \times \mathcal{T}$, $\mathcal{O}_{\mathrm{TD}}$ returns $\mathsf{KI.Trapdoor}(\mathsf{pk}, \mathsf{sk}_i, w)$ if $(w, i) \notin \{(w_0^*, i^*), (w_1^*, i^*)\}$. Otherwise, it returns $\perp$.

Definition 8 (IND-KI-CKA) *A KU-PEKS scheme Π_{KI} is said to be* IND-KI-CKA *secure if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{KI\text{-}CKA}}_{\Pi_{\mathrm{KI}},\mathcal{A}}(1^\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{KI\text{-}CKA}}_{\Pi_{\mathrm{KI}},\mathcal{A}}(1^\lambda) = 1] - 1/2|$ is negligible in λ.*

Next, we define KI-Computational Consistency. For any PPT adversary $\mathcal{A}$, we consider the following game.

$\mathsf{Exp}^{\mathsf{KI\text{-}Cons}}_{\Pi_{\mathrm{KI}},\mathcal{A}}(1^\lambda)$
$(\mathsf{pk}, \mathsf{sk}_0, \mathsf{hk}) \leftarrow \mathsf{KI.Setup}(1^\lambda)$
$(w_0^*, w_1^*, i^*, j^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{KL}}, \mathcal{O}_{\mathrm{RK}}}(\mathsf{pk})$ s.t. $|w_0^*| = |w_1^*|$
$\mathsf{c}_{w_0^*, i^*} \leftarrow \mathsf{KI.Enc}(\mathsf{pk}, w_0^*, i^*)$,
$(\mathsf{sk}_{j^*}, \mathsf{rk}_{j^*}) \leftarrow \mathsf{KI.Upd}(\mathsf{sk}_0, \Delta\text{-Gen}(\mathsf{pk}, \mathsf{hk}, j^*))$
$\mathsf{td}_{w_1^*, j^*} \leftarrow \mathsf{KI.Trapdoor}(\mathsf{pk}, \mathsf{sk}_{j^*}, w_1^*)$
if $\mathsf{KI.Test}(\mathsf{td}_{w_1^*, j^*}, \mathsf{c}_{w_0^*, i^*}) = 1 \wedge\ w_0^* \neq w_1^*$ **return** 1
else return 0

$\mathcal{A}$ can access $\mathcal{O}_{\mathrm{KL}}$ and $\mathcal{O}_{\mathrm{RK}}$, which are the same as those in the IND-KI-CKA game.

[3] For simplicity, we assume $\mathcal{A}$ issues $i \in \mathcal{T}$ to $\mathcal{O}_{\mathrm{RK}}$ after $\mathcal{A}$ issues i to $\mathcal{O}_{\mathrm{KL}}$ except $\mathrm{L} = \{\star\}$ (i.e., $\mathcal{A}$ obtains hk from $\mathcal{O}_{\mathrm{KL}}$).

Definition 9 (KI-Computational Consistency) *A KU-PEKS scheme Π_{KI} is said to meet* KI-Computational Consistency *if for all PPT adversaries $\mathcal{A}$, its advantage* $\mathsf{Adv}^{\mathsf{KI\text{-}Cons}}_{\Pi_{\mathrm{KI}},\mathcal{A}}(1^\lambda) := \Pr[\mathsf{Exp}^{\mathsf{KI\text{-}Cons}}_{\Pi_{\mathrm{KI}},\mathcal{A}}(1^\lambda) = 1]$ *is negligible in λ.*

4.2 Building Block: Anonymous Key-insulated IBE for Master Keys

We take a similar strategy for the key-insulation model as in Abdalla et al.'s work [2], which showed the transformation from an anonymous IBE scheme to a PEKS scheme. Namely, we consider a transformation from an anonymous IBE scheme with certain key-insulated functionality to a KU-PEKS scheme (in the key-insulation model). Key-insulated IBE (KI-IBE, or IKE for short) [20, 30] is a promising candidate, however, (i) the existing scheme is not anonymous, and (ii) the key-insulated functionality is insufficient to realize key-insulated functionality of KU-PEKS. Let us elaborate the latter. In the Abdalla et al. transformation, a master key of an IBE scheme turns to be a secret key of the resulting PEKS scheme, and secret keys of the IBE scheme are used as trapdoors of the PEKS scheme. However, the existing IKE schemes [20,30] have key-insulated functionality for *users' secret keys*. Therefore, if we apply the the Abdalla et al. transformation to those IKE schemes, then we just get a PEKS scheme with key-insulated functionality for *trapdoors*. Actually, Emura et al. [18] applied the Abdalla et al. transformation from a revocable IBE scheme [6,24,27], which is an IBE enabling the central authority to efficiently revoke secret keys, to a PEKS scheme with revocation functionality for trapdoors.

Thus, we here introduce a new key-insulated cryptographic primitive, *IKE for master keys* (MIKE for short). Roughly speaking, MIKE captures leakage of the master key, whereas IKE focuses on leakage of users' secret keys. This primitive may be of independent interest. We also consider the anonymity of MIKE since Abdalla et al. transformation requires the underlying IBE scheme to be anonymous.

A MIKE scheme $\mathcal{MIKE}$ consists of six-tuple algorithms $(\mathsf{Init}, \mathsf{UpdGen}, \mathsf{MKUpd}, \mathsf{KG}, \mathsf{IBEnc}, \mathsf{IBDec})$ defined as follows.

- $\mathsf{Init}(1^\lambda) \to (\mathsf{prms}, \mathsf{mk}_0, \mathsf{mhk})$: given the security parameter 1^λ, it outputs a public parameter prms, an initial master secret key mk_0, and a master helper key mhk.
- $\mathsf{UpdGen}(\mathsf{prms}, \mathsf{mhk}, \mathtt{T}) \to \mathsf{up}_\mathtt{T}$: given prms, mhk, and a time period $\mathtt{T} \in \mathcal{T}$, it outputs update information $\mathsf{up}_\mathtt{T}$ for $\mathtt{T}$.
- $\mathsf{MKUpd}(\mathsf{prms}, \mathsf{mk}_{\mathtt{T}'}, \mathsf{up}_\mathtt{T}) \to \mathsf{mk}_\mathtt{T}$: given prms, $\mathsf{mk}_{\mathtt{T}'}$, and $\mathsf{up}_\mathtt{T}$, it outputs an updated master key $\mathsf{mk}_\mathtt{T}$.
- $\mathsf{KG}(\mathsf{prms}, \mathsf{mk}_\mathtt{T}, \mathtt{I}) \to \mathsf{dk}_{\mathtt{T},\mathtt{I}}$: given prms, $\mathsf{mk}_\mathtt{T}$, and an identity $\mathtt{I} \in \mathcal{I}$, it outputs a decryption key $\mathsf{dk}_{\mathtt{T},\mathtt{I}}$ for $\mathtt{I}$ and the time period $\mathtt{T}$.
- $\mathsf{IBEnc}(\mathsf{prms}, m, \mathtt{T}, \mathtt{I}) \to \mathsf{ct}_{\mathtt{T},\mathtt{I}}$: given prms, a plaintext $m \in \mathcal{M}$, a current time period $\mathtt{T}$, and $\mathtt{I} \in \mathcal{I}$, it outputs a ciphertext $\mathsf{ct}_{\mathtt{T},\mathtt{I}}$.
- $\mathsf{IBDec}(\mathsf{prms}, \mathsf{dk}_{\mathtt{T},\mathtt{I}}, \mathsf{ct}_{\mathtt{T},\mathtt{I}}) \to m$ or $\perp$: given prms, $\mathsf{dk}_{\mathtt{T},\mathtt{I}}$, and $\mathsf{ct}_{\mathtt{T},\mathtt{I}}$, it outputs m or $\perp$.

We require $\mathcal{MIKE}$ meets the following correctness property: For all $\lambda \in \mathbb{N}$, all $(\mathsf{prms}, \mathsf{mk}_0, \mathsf{mhk}) \leftarrow \mathsf{Init}(1^\lambda)$, all $M \in \mathcal{M}$, all $\mathtt{I} \in \mathcal{I}$, all $\mathtt{T}, \mathtt{T}' \in \mathcal{T}$, it holds that $M \leftarrow \mathsf{IBDec}(\mathsf{prms}, \mathsf{KG}(\mathsf{prms}, \mathsf{MKUpd}(\mathsf{prms}, \mathsf{mk}_{\mathtt{T}'}, \mathsf{UpdGen}(\mathsf{prms}, \mathsf{mhk}, \mathtt{T})), \mathtt{I}), \mathsf{IBEnc}(\mathsf{prms}, M, \mathtt{T}, \mathtt{I}))$.

Security. We consider two kinds of security notions of MIKE: indistinguishability against key exposure and chosen plaintext attacks for MIKE (IND-ID-KI-CPA) and anonymity (ANO-ID-KI-CPA).

First, we define IND-ID-KI-CPA security. Let $\mathcal{A}$ be a PPT adversary, and we consider the following experiment.

$\mathsf{Exp}^{\mathsf{ID\text{-}KI\text{-}CPA}}_{\mathcal{MIKE},\mathcal{A}}(1^\lambda)$

$(\mathsf{prms}, \mathsf{mk}_0, \mathsf{mhk}) \leftarrow \mathsf{Init}(1^\lambda)$

$(m_0^*, m_1^*, \mathtt{T}^*, \mathtt{I}^*, \mathit{state}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{EXT}}, \mathcal{O}_{\mathrm{LEAK}}}(\mathsf{prms})$ s.t. $|m_0^*| = |m_1^*|$

$b \xleftarrow{\$} \{0,1\}$, $\mathsf{ct}_{\mathtt{T}^*,\mathtt{I}^*} \leftarrow \mathsf{IBEnc}(\mathsf{prms}, m_b^*, \mathtt{T}^*, \mathtt{I}^*)$

$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{EXT}}, \mathcal{O}_{\mathrm{LEAK}}}(\mathit{state}, \mathsf{ct}_{\mathtt{T}^*,\mathtt{I}^*})$

if $b' = b$ **return** 1 **else return** 0

$\mathcal{A}$ can access the following two oracles $\mathcal{O}_{\mathrm{EXT}}$ and $\mathcal{O}_{\mathrm{LEAK}}$, which are defined as follows.

$\mathcal{O}_{\mathrm{EXT}}$: For a query $(\mathtt{T}, \mathtt{I}) \in \mathcal{T} \times \mathcal{I}$ from $\mathcal{A}$, $\mathcal{O}_{\mathrm{EXT}}$ recalls $\mathsf{mk}_\mathtt{T}$ if it is already generated. Otherwise, $\mathcal{O}_{\mathrm{EXT}}$ computes $\mathsf{mk}_\mathtt{T} \leftarrow \mathsf{MKUpd}(\mathsf{mk}_0, \mathsf{UpdGen}(\mathsf{mhk}, \mathtt{T}))$, and stores it. $\mathcal{O}_{\mathrm{EXT}}$ then returns $\mathsf{KG}(\mathsf{mk}_\mathtt{T}, \mathtt{I})$ if $(\mathtt{T}, \mathtt{I}) \neq (\mathtt{T}^*, \mathtt{I}^*)$ in $\mathsf{Exp}^{\mathsf{ID\text{-}KI\text{-}CPA}}_{\mathcal{MIKE},\mathcal{A}}(1^\lambda)$.

$\mathcal{O}_{\mathrm{LEAK}}$: Let $\mathcal{L} := \emptyset$ be an initial list. For a query $\mathtt{T} \in \mathcal{T} \cup \{\star\}$, $\mathcal{O}_{\mathrm{LEAK}}$ returns $\mathsf{mk}_\mathtt{T}$ if $\mathtt{T} \notin \mathcal{T} \backslash \{\mathtt{T}^*\}$ and $\star \notin \mathcal{L}$, and adds $\mathtt{T}$ to $\mathcal{L}$.[4] Else if $\mathtt{T} = \star$ and $\mathcal{L} = \emptyset$, $\mathcal{O}_{\mathrm{LEAK}}$ returns mhk, and adds $\star$ to $\mathcal{L}$. Otherwise, $\mathcal{O}_{\mathrm{LEAK}}$ returns $\perp$.

Definition 10 (IND-ID-KI-CPA) *$\mathcal{MIKE}$ is said to be* IND-ID-KI-CPA *secure if for all PPT adversaries $\mathcal{A}$,* $\mathsf{Adv}^{\mathsf{ID\text{-}KI\text{-}CPA}}_{\mathcal{MIKE},\mathcal{A}}(1^\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{ID\text{-}KI\text{-}CPA}}_{\mathcal{MIKE},\mathcal{A}}(1^\lambda) = 1] - 1/2|$ *is negligible in λ.*

We next define ANO-ID-KI-CPA security. We consider the following experiment for any PPT adversary $\mathcal{A}$.

[4] If $\mathsf{mk}_\mathtt{T}$ is not stored, $\mathcal{O}_{\mathrm{LEAK}}$ generates it by $\mathsf{MKUpd}(\mathsf{mk}_0, \mathsf{UpdGen}(\mathsf{mhk}, \mathtt{T}))$ and stored it.

$\mathsf{Exp}^{\mathsf{ANO\text{-}KI\text{-}CPA}}_{\mathcal{MIKE},\mathcal{A}}(1^\lambda)$

$(\mathsf{prms}, \mathsf{mk}_0, \mathsf{mhk}) \leftarrow \mathsf{Init}(1^\lambda)$

$(m^*, \mathtt{T}^*, \mathtt{I}_0^*, \mathtt{I}_1^*, state) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{EXT}}, \mathcal{O}_{\mathrm{LEAK}}}(\mathsf{prms})$

$b \xleftarrow{\$} \{0,1\}, \quad \mathsf{ct}_{\mathtt{T}^*, \mathtt{I}_b^*} \leftarrow \mathsf{IBEnc}(\mathsf{prms}, m^*, \mathtt{T}^*, \mathtt{I}_b^*)$

$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{EXT}}, \mathcal{O}_{\mathrm{LEAK}}}(state, \mathsf{ct}_{\mathtt{T}^*, \mathtt{I}_b^*})$

if $b' = b$ **return** 1 **else return** 0

$\mathcal{A}$ can access the same oracles as those in the IND-ID-KI-CPA game. Note that the restriction of $\mathcal{O}_{\mathrm{EXT}}$ should be changed as follows: $\mathcal{O}_{\mathrm{EXT}}$ returns $\mathsf{KG}(\mathsf{mk}_\mathtt{T}, \mathtt{I})$ if $(\mathtt{T}, \mathtt{I}) \in \{(\mathtt{T}^*, \mathtt{I}_0^*), (\mathtt{T}^*, \mathtt{I}_1^*)\}$.

Definition 11 (ANO-ID-KI-CPA) *$\mathcal{MIKE}$ is said to be* ANO-ID-KI-CPA *secure if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ANO\text{-}KI\text{-}CPA}}_{\mathcal{MIKE},\mathcal{A}}(1^\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{ANO\text{-}KI\text{-}CPA}}_{\mathcal{MIKE},\mathcal{A}}(1^\lambda) = 1] - 1/2|$ is negligible in λ.*

Construction. We will give a concrete construction of this new primitive from the symmetric external Diffie-Hellman (SXDH) assumption (without random oracles) in Section 6.

4.3 Generic Construction from KI-PKE and MIKE

In this section, we show a generic construction of a KU-PEKS scheme Π_{KI} in the key-insulation model from any KI-PKE scheme $\mathcal{KIE}$, any anonymous MIKE scheme $\mathcal{MIKE}$, and any collusion-resistant hash function family $\mathcal{H}$. Basically, we can construct Π_{KI} in a similar way to the generic construction of Π_{KE} in Section 3.2. However, the construction only achieves *sequential key updates*, that is, a re-encryption key rk_i for $i \in \mathcal{T}$ can be used for only updating a ciphertext $\mathsf{c}_{w,i-1}$ encrypted during the previous period $i-1 \in \mathcal{T}$. To achieve random access key updates, i.e., to realize update of a ciphertext $\mathsf{c}_{w,j}$ for a time period $j \in \mathcal{T}$ to any time period $i \in \mathcal{T}$, we adopt the *KUNode algorithm* (or, the *complete subtree (CS) method*), which was used for broadcast encryption [25], revocable IBE [6], and so forth. The KUNode algorithm is usually used for efficiently revoking malicious users, whereas we would like to use it to efficiently realize random access updates. Therefore, we modify the KUNode algorithm to fit our purpose as follows (see [25,6] for the original KUNode algorithm).

The modified KUNode algorithm. Let BTGen be an algorithm that takes N as input, and outputs a binary tree BT with N leaves, where N is a power of two for simplicity. Each time period $i \in \mathcal{T}$ is assigned to a leaf node, and the corresponding i-th leaf node is denoted by η_i. For the sake of simplicity, we assume $N = |\mathcal{T}|$. Now the depth of BT is $\log|\mathcal{T}| + 1$, and the number of all nodes is $2^{\log|\mathcal{T}|+1} - 1 = 2|\mathcal{T}| - 1$. $\mathsf{Path}(\mathtt{BT}, \eta_i)$ denotes a set of nodes on the path from a root node to η_i. Note that it includes the root node and η_i. The modified $\mathsf{KUNode}(\mathtt{BT}, i)$ algorithm takes as input the binary tree BT and a time period $i \in \mathcal{T}$, and outputs a set of nodes. The modified $\mathsf{KUNode}(\mathtt{BT}, i)$ algorithm is executed as follows. It sets $\mathcal{X} := \emptyset$. For each non-leaf node $\theta \in \mathsf{Path}(\mathtt{BT}, \eta_i)$, it adds the left child θ_L of θ to $\mathcal{X}$ if $\theta_L \notin \mathsf{Path}(\mathtt{BT}, \eta_i)$. Finally, it outputs $\mathcal{X}$. Note that the size of $\mathcal{X}$ is $O(\log|\mathcal{T}|)$.

We are ready to show our construction. Let $\mathcal{KIE} = (\mathsf{KIKG}, \mathsf{KIUG}, \mathsf{KIU}, \mathsf{KIE}, \mathsf{KID})$ and $\mathcal{MIKE} = (\mathsf{Init}, \mathsf{UpdGen}, \mathsf{MKUpd}, \mathsf{KG}, \mathsf{IBEnc}, \mathsf{IBDec})$ be a KI-PKE scheme with a set of time periods $\widehat{\mathcal{T}}$ such that $|\widehat{\mathcal{T}}| = 2|\mathcal{T}| - 1$ and a MIKE scheme with $\mathcal{T}$, respectively. Our construction of Π_{KI} is given in Fig. 2. In this construction, we consider the following function $\mathsf{Lab} : i \in \mathcal{T} \mapsto i + |\mathcal{T}| - 1 \in \mathbb{Z}$ for the modified KUNode algorithm. First, we label each node of BT as θ_i $(1 \le i \le 2|\mathcal{T}| - 1)$ from the root node. Hence, the root node is θ_1 and leaf nodes are $\theta_{|\mathcal{T}|}, \ldots, \theta_{2|\mathcal{T}|-1}$. Then, each time period $i \in \mathcal{T}$ is stored in a leaf node $\theta_{\mathsf{Lab}(i)}$, and we write $\eta_i := \theta_{\mathsf{Lab}(i)}$. Moreover, for readability, $\mathsf{Path}(\mathtt{BT}, \eta_i)$ and $\mathsf{KUNode}(\mathtt{BT}, i)$ are regarded as a set of indices of the corresponding nodes. Namely, we write $\{1, j_1, j_2, \ldots, \mathsf{Lab}(i)\} = \mathsf{Path}(\mathtt{BT}, \eta_i)$ and $\{h_1, h_2, \ldots, h_k\} = \mathsf{KUNode}(\mathtt{BT}, i)$, instead of $\{\theta_1, \theta_{j_1}, \theta_{j_2}, \ldots, \theta_{\mathsf{Lab}(i)} (= \eta_i)\} = \mathsf{Path}(\mathtt{BT}, \eta_i)$ and $\{\theta_{h_1}, \theta_{h_2}, \ldots, \theta_{h_k}\} = \mathsf{KUNode}(\mathtt{BT}, i)$, respectively.

We obtain the following theorem, and omit the proof since it can be proved in a way similar to Theorem 1 and the Abdalla et al.'s transformation [2].

Theorem 2 *If $\mathcal{KIE}$ is* IND-KI-CPA *secure, $\mathcal{MIKE}$ is* IND-ID-KI-CPA *secure and* ANO-ID-KI-CPA *secure, and $\mathcal{H}$ satisfies* Collision Resistance*, the proposed construction given in Fig. 2 is* IND-KI-CKA *secure and meets* KI-Computational Consistency*.*

5 Implementation Results

In this section, we provide the implementation results of the following three instantiations. The first two are instantiations of the key-evolution model, and the last one is one of the key-insulation model:

1. $\Pi^{\mathsf{rom}}_{\mathrm{KE}}$: An instantiation of the key-evolution model with ElGamal PKE [17] and Boneh-Franklin IBE [8], which is the most popular (anonymous) IBE scheme secure in the random oracle model.
2. $\Pi^{\mathsf{std}}_{\mathrm{KE}}$: An instantiation of the key-evolution model with ElGamal PKE [17] and Jutla-Roy IBE [21], which is the most efficient anonymous IBE scheme without random oracles in terms of parameter sizes.

$\underline{\mathsf{KI.Setup}(1^\lambda)}$:
$\mathtt{BT} \leftarrow \mathsf{BTGen}(|\mathcal{T}|)$
$(\mathsf{EK}, \mathsf{DK}_0, \mathsf{HK}) \leftarrow \mathsf{KIKG}(1^\lambda)$
$(\mathsf{prms}, \mathsf{mk}_0, \mathsf{mhk}) \leftarrow \mathsf{Init}(1^\lambda)$
$\mathsf{H} \xleftarrow{\$} \mathcal{H}$
$\mathsf{pk} := (\mathtt{BT}, \mathsf{EK}, \mathsf{prms}, \mathsf{H})$
$\mathsf{sk}_0 := (\mathsf{DK}_0, \mathsf{mk}_0)$
$\mathsf{hk} := (\mathsf{HK}, \mathsf{mhk})$
return $(\mathsf{pk}, \mathsf{sk}_0, \mathsf{hk})$

$\underline{\Delta\text{-}\mathsf{Gen}(\mathsf{pk}, \mathsf{hk}, i)}$:
parse $\mathsf{pk} = (\mathtt{BT}, \mathsf{EK}, \mathsf{prms}, \mathsf{H})$
parse $\mathsf{hk} = (\mathsf{HK}, \mathsf{mhk})$
for $\forall \ell \in \mathsf{KUNode}(\mathtt{BT}, i)$
  $\mathsf{UP}_\ell \leftarrow \mathsf{KIUG}(\mathsf{HK}, \ell)$
$\mathsf{up}_i \leftarrow \mathsf{UpdGen}(\mathsf{prms}, \mathsf{mhk}, i)$
$\delta_i := (\{\mathsf{UP}_\ell\}_{\ell \in \mathsf{KUNode}(\mathtt{BT}, i)}, \mathsf{up}_i)$
return δ_i

$\underline{\mathsf{KI.Upd}(\mathsf{pk}, \mathsf{sk}_{i'}, \delta_i)}$:
parse $\mathsf{pk} = (\mathtt{BT}, \mathsf{EK}, \mathsf{prms}, \mathsf{H})$
parse $\mathsf{sk}_{i'} = (\mathsf{DK}_0, \mathsf{mk}_{i'})$
parse $\delta_i = (\{\mathsf{UP}_\ell\}_{\ell \in \mathsf{KUNode}(\mathtt{BT}, i)}, \mathsf{up}_i)$
for $\forall \ell \in \mathsf{KUNode}(\mathtt{BT}, i)$
  $\mathsf{DK}_\ell \leftarrow \mathsf{KIUG}(\mathsf{DK}_0, \mathsf{UP}_\ell)$
$\mathsf{mk}_i \leftarrow \mathsf{MKUpd}(\mathsf{prms}, \mathsf{mk}_{i'}, \mathsf{up}_i)$
$\mathsf{sk}_i := (\mathsf{DK}_0, \mathsf{mk}_i)$
$\mathsf{rk}_i := (\{\mathsf{DK}_\ell\}_{\ell \in \mathsf{KUNode}(\mathtt{BT}, i)})$
return $(\mathsf{sk}_i, \mathsf{rk}_i)$

$\underline{\mathsf{KI.Enc}(\mathsf{pk}, w, i)}$:
parse $\mathsf{pk} = (\mathtt{BT}, \mathsf{EK}, \mathsf{prms}, \mathsf{H})$
for $\forall \ell \in \mathsf{Path}(\mathtt{BT}, \theta_{\mathsf{Lab}(i)}) \setminus \{1\}$
// Do except for the root node θ_1
  $\mathsf{ct}_\ell \leftarrow \mathsf{KIE}(\mathsf{EK}, \mathsf{H}(w), \ell)$
$R \xleftarrow{\$} \mathcal{M}$
// $\mathcal{M}$: the plaintext space of $\mathcal{MIKE}$
$\mathsf{ct}_{i,w} \leftarrow \mathsf{IBEnc}(\mathsf{prms}, R, i, \mathsf{H}(w))$
$\mathsf{c}_{w,i} := (R, \{\mathsf{ct}_\ell\}_{\ell \in \mathsf{Path}(\mathtt{BT}, \theta_{\mathsf{Lab}(i)})}, \mathsf{ct}_{i,w})$
return $\mathsf{c}_{w,i}$

$\underline{\mathsf{KI.ReEnc}(\mathsf{pk}, \mathsf{rk}_i, \mathsf{c}_{w,j})}$:
parse $\mathsf{rk}_i = (\{\mathsf{DK}_\ell\}_{\ell \in \Theta_i})$
// $\Theta_i = \mathsf{KUNode}(\mathtt{BT}, i)$
parse $\mathsf{c}_{w,j} = (R, \{\mathsf{ct}_\ell\}_{\ell \in \Theta_j}, \mathsf{ct}_{j,w})$
// $\Theta_j = \mathsf{Path}(\mathtt{BT}, \theta_{\mathsf{Lab}(j)})$
if $\Theta_i \cap \Theta_j = \emptyset$
// It occurs if and only if $i \leq j$
  return $\perp$
else
  $\{\ell^*\} := \Theta_i \cap \Theta_j$
  // It contains exactly one element
  $\mathsf{H}(w) \leftarrow \mathsf{KID}(\mathsf{DK}_{\ell^*}, \mathsf{ct}_{\ell^*})$
  // Do the same procedure of KI.Enc
  for $\forall \ell \in \mathsf{Path}(\mathtt{BT}, \theta_{\mathsf{Lab}(i)}) \setminus \{1\}$
    $\mathsf{ct}_\ell \leftarrow \mathsf{KIE}(\mathsf{EK}, \mathsf{H}(w), \ell)$
  $R \xleftarrow{\$} \mathcal{M}$
  $\mathsf{ct}_{i,w} \leftarrow \mathsf{IBEnc}(\mathsf{prms}, R, i, \mathsf{H}(w))$
  $\mathsf{c}_{w,i} := (R, \{\mathsf{ct}_\ell\}_{\ell \in \mathsf{Path}(\mathtt{BT}, \theta_{\mathsf{Lab}(i)})}, \mathsf{ct}_{i,w})$
  return $\mathsf{c}_{w,i}$

$\underline{\mathsf{KI.Trapdoor}(\mathsf{pk}, \mathsf{sk}_i, w')}$:
parse $\mathsf{pk} = (\mathtt{BT}, \mathsf{EK}, \mathsf{prms}, \mathsf{H})$
parse $\mathsf{sk}_i = (\mathsf{DK}_0, \{\mathsf{DK}_\ell\}_{\ell \in \Theta_i}, \mathsf{mk}_i)$
$\mathsf{dk}_{i,w'} \leftarrow \mathsf{KG}(\mathsf{prms}, \mathsf{mk}_i, \mathsf{H}(w'))$
$\mathsf{t}_{w',i} := \mathsf{dk}_{i,w'}$
return $\mathsf{t}_{w',i}$

$\underline{\mathsf{KI.Test}(\mathsf{pk}, \mathsf{t}_{w',i}, \mathsf{c}_{w,j})}$:
parse $\mathsf{pk} = (\mathtt{BT}, \mathsf{EK}, \mathsf{prms}, \mathsf{H})$
parse $\mathsf{c}_{w,j} = (\{R, \mathsf{ct}_\ell\}_{\ell \in \Theta_j}, \mathsf{ct}_{j,w})$
if $i \neq j$
  return 0
else if $R = \mathsf{IBDec}(\mathsf{prms}, \mathsf{t}_{w',i}, \mathsf{ct}_{j,w})$
  return 1
else if $R \neq \mathsf{IBDec}(\mathsf{prms}, \mathsf{t}_{w',i}, \mathsf{ct}_{j,w})$
  return 0

Fig. 2 A generic construction of Π_{KI} from $\mathcal{KIE}$, $\mathcal{MIKE}$, and $\mathcal{H}$.

Table 1 Runnning time of core algorithms (unit: msec)

	Setup	Upd	Enc	ReEnc	Trapdoor	Test
$\Pi_{\mathrm{KE}}^{\mathsf{rom}}$	60.3	57.2	93.4	97.3	3.7	31.2
$\Pi_{\mathrm{KE}}^{\mathsf{std}}$	151.5	147.8	72.3	72.2	297.6	165.8
$\Pi_{\mathrm{KI}}^{\mathsf{std}}$	27,775.9	28,940.2	522.1	707.1	144.8	131.7

3. $\Pi_{\mathrm{KI}}^{\mathsf{std}}$: An instantiation of the key-insulation model with Watanabe-Shikata KI-PKE [30], which is the most efficient KI-PKE scheme, and a direct construction of an anonymous MIKE scheme, which will be provided in section 6. We set $t := \log |\mathcal{T}| = 7$ since we suppose that the key is updated once a month for ten years.

All the instantiations are secure under the simple assumptions, and we use SHA-256 as the underlying hash function in each instantiation. The first one achieves the most efficient parameters, though the security relies on random oracles. The third one is less efficient than the other two, however it does not require to update public keys.

Table 1 and Fig. 3 show the running time of core algorithms in each instantiation on Raspberry Pi (unit: msec). Raspberry Pi is a small single-board computer, which seems a great platform for building IoT devices. Specifically, we employ Raspberry Pi 3 model B with Broadcom BCM2837 (1.2 GHz) processor, 1 GB RAM, and the Linux (Raspbian 9.1) OS. We also use the following libraries: TEPLA 2.0 and GMP 5.1.3. TEPLA, which is an open source C encryption library developed by University of Tsukuba, Japan, includes elliptic curve computations and pairing functions. Since the running time of $\Pi_{\mathrm{KI}}^{\mathsf{std}}$ varies depending on time period i, we measure ten times randomly and obtain the average. All the algorithms except for Setup, Upd, and ReEnc in $\Pi_{\mathrm{KI}}^{\mathsf{std}}$ can be executed in 550 msec, and hence our proposals seem acceptable to use on IoT devices. Note that though Setup and Upd require more than 25 sec, they are not relatively frequently executed.

Table 2 shows comparisons of parameter size among three instantiations. $\Pi_{\mathrm{KE}}^{\mathsf{rom}}$ and $\Pi_{\mathrm{KE}}^{\mathsf{std}}$ are more efficient

Table 2 Comparison of parameter size among instantiations of the proposed schemes. #pk, #sk, #rk, #td, and #c denote the sizes of public keys, secret keys, re-encryption keys, trapdoors, and ciphertexts, respectively. $[a, b, c, d]$ means that the parameter contains a elements of $\mathbb{Z}_p$, b elements of $\mathbb{G}_1$, c elements of $\mathbb{G}_2$, and d elements of $\mathbb{G}_T$. We set $t := \log|\mathcal{T}|$. We assume the plaintext space of the underlying Bone-Franklin IBE in $\Pi^{\mathsf{rom}}_{\mathrm{KE}}$ is $\mathbb{Z}_p$. Asmp. stands for assumptions, and DDH1, DBDH, and SXDH denote the decisional Diffie-Hellman, decisional bilinear Diffie-Hellman, and symmetric external Diffie-Hellman assumptions, respectively (see Section 6 and Appendix C for formal definitions of them).

	Update pk?	Update counter management	#pk	#sk	#rk	#td	#c	Asmp.
$\Pi^{\mathsf{rom}}_{\mathrm{KE}}$	Yes	For each user	$[0,4,0,0]$	$[2,0,0,0]$	$[1,0,0,0]$	$[0,1,0,0]$	$[2,3,0,0]$	DDH1, DBDH
$\Pi^{\mathsf{std}}_{\mathrm{KE}}$	Yes	For each user	$[0,7,0,1]$	$[9,0,1,0]$	$[1,0,0,0]$	$[0,0,5,0]$	$[1,5,0,1]$	SXDH
$\Pi^{\mathsf{std}}_{\mathrm{KI}}$	No	global $\mathcal{T}$	$[0,13,7,1]$	$[8,0,17,0]$	$[0,0,O(t),0]$	$[0,0,5,0]$	$[2t+1,3t+3,0,t+1]$	SXDH

than $\Pi^{\mathsf{std}}_{\mathrm{KI}}$ since the size of ciphertexts in $\Pi^{\mathsf{std}}_{\mathrm{KI}}$ increases proportionally to $t := \log|\mathcal{T}|$, where $\mathcal{T}$ is the global time-period set. However, $\Pi^{\mathsf{std}}_{\mathrm{KI}}$ can keep the public key fixed during the protocol. Moreover, when we suppose multi-user model, the server only manages *global* $\mathcal{T}$ in the key-insulation model, while it has to deal with multiple time-period sets to manage the updating counter for each user in the key-evolution model. In this sense, the key-insulation model is more practical than the key-evolution model.

6 Direct Construction of Anonymous MIKE

We construct an anonymous MIKE scheme $\mathcal{MIKE} = (\mathsf{Init}, \mathsf{UpdGen}, \mathsf{MKUpd}, \mathsf{IBEnc}, \mathsf{IBDec})$ over asymmetric bilinear groups of prime order.

Bilinear Groups. Let p be a prime, $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be multiplicative cyclic groups of order p, and g_1 and g_2 be (random) generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. We consider an efficiently computable and non-degenerate bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following bilinear property: for any $u, u' \in \mathbb{G}_1$ and $v, v' \in \mathbb{G}_2$, $e(uu', v) = e(u, v)e(u', v)$ and $e(u, vv') = e(u, v)e(u, v')$. The bilinear map e is called symmetric or a "Type-1" pairing if $\mathbb{G}_1 = \mathbb{G}_2$. Otherwise, it is called asymmetric. In the asymmetric setting, e is called a "Type-2" pairing if there is an efficiently computable isomorphism either from $\mathbb{G}_1$ to $\mathbb{G}_2$ or from $\mathbb{G}_2$ to $\mathbb{G}_1$. If no efficiently computable isomorphisms are known, then it is called a "Type-3" pairing. We focus on the Type-3 pairing for our construction. In our construction, we employ a bilinear group generator $\mathcal{G}$, which is an algorithm that takes the security parameter 1^λ as input and outputs a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$.

Construction. This construction is based on the existing IKE scheme [30], however we give it a twist since we have to consider both the exposure of master keys and the anonymity. As a result, the proposed construction is more complicated than the existing scheme.

- $\mathsf{Init}(1^\lambda)$: Run $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathcal{G}(1^\lambda)$. Choose $x_i, y_i, \beta_i, \beta'_i \xleftarrow{\$} \mathbb{Z}_p$ with $0 \le i \le 4$ and $\alpha \xleftarrow{\$} \mathbb{Z}_p^\times$, and set

  $z = e(g_1, g_2)^{x_0\alpha - y_0}$, $u_1 := g_1^{x_1\alpha - y_1}$, $w_1 := g_1^{x_2\alpha - y_2}$,
  $h_1 := g_1^{x_3\alpha - y_3}$, $v_1 := g_1^{x_4\alpha - y_4}$,
  $B_{x,i} := g_2^{x_i + \beta_i}$ for every $i \in \{0, 1, \ldots, 4\}$,
  $B_{y,i} := g_2^{-y_i - \beta'_i}$ for every $i \in \{0, 1, \ldots, 4\}$.

  Output

  $\mathsf{prms} := (g_1, g_1^\alpha, u_1, w_1, h_1, v_1, z)$,
  $\mathsf{mk}_0 := (g_2, \{\beta_i, \beta'_i\}_{i=0}^4)$,
  $\mathsf{mhk} := (g_2, \{B_{x,i}, B_{y,i}\}_{i=0}^4)$.

- $\mathsf{UpdGen}(\mathsf{prms}, \mathsf{mhk}, \mathtt{T})$: Choose $s_1, s_2 \xleftarrow{\$} \mathbb{Z}_p$. For $i \in \{1, 2\}$, compute

  $D_i := g_2^{s_i}$,
  $D_{x,i} := B_{x,2}^{s_i} = g_2^{s_i(x_2 + \beta_2)}$,
  $D'_{x,1} := B_{x,0}\left(B_{x,1}^{\mathtt{T}} B_{x,3}\right)^{s_1}$
  $= g_2^{x_0 + s_1(x_1\mathtt{T} + x_3) + \beta_0 + s_1(\beta_1\mathtt{T} + \beta_3)}$,
  $D'_{x,2} := \left(B_{x,1}^{\mathtt{T}} B_{x,3}\right)^{s_2}$
  $= g_2^{s_2(x_1\mathtt{T} + x_3) + s_2(\beta_1\mathtt{T} + \beta_3)}$,
  $D''_{y,i} := B_{x,4}^{s_i} = g_2^{s_i(x_4 + \beta_4)}$,
  $D_{y,i} := B_{y,2}^{s_i} = g_2^{-s_i(y_2 + \beta'_2)}$,
  $D'_{y,1} := B_{y,0}\left(B_{y,1}^{\mathtt{T}} B_{y,3}\right)^{s_1}$
  $= g_2^{-y_0 - s_1(y_1\mathtt{T} + y_3) - \beta'_0 - s_1(\beta'_1\mathtt{T} + \beta'_3)}$,
  $D'_{y,2} := \left(B_{y,1}^{\mathtt{T}} B_{y,3}\right)^{s_2} = g_2^{-s_2(y_1\mathtt{T} + y_3) - s_2(\beta'_1\mathtt{T} + \beta'_3)}$,
  $D''_{y,i} := B_{y,4}^{s_i} = g_2^{-s_i(y_4 + \beta'_4)}$.

  Output

  $\mathsf{up}_{\mathtt{T}} := (\{D_i, D_{x,i}, D'_{x,i}, D''_{x,i}, D_{y,i}, D'_{y,i}, D''_{y,i}\}_{i=1}^2)$.

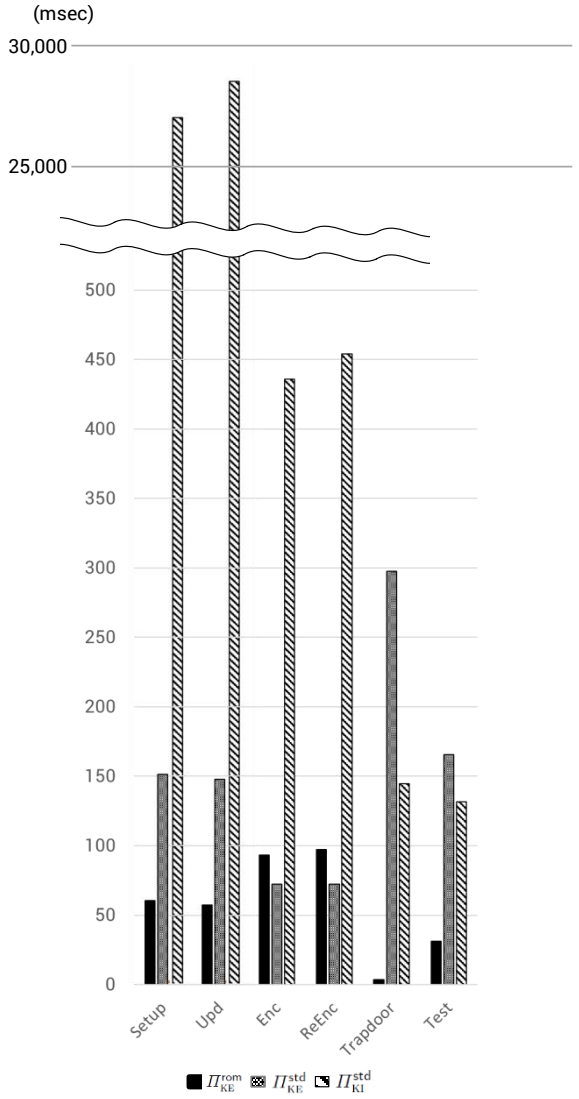

Fig. 3 Comparison among instantiations of the proposed schemes (unit: msec)

- $\mathsf{MKUpd}(\mathsf{prms}, \mathsf{mk}_{\mathtt{T}'}, \mathsf{up}_{\mathtt{T}})$: For $i \in \{1,2\}$, compute

$$\begin{aligned}
&\mathtt{D}_i := D_i,\\
&\mathtt{D}_{x,i} := D_{x,i} D_i^{-\beta_2} = g_2^{s_i x_2},\\
&\mathtt{D}'_{x,1} := D'_{x,1} g_2^{-\beta_0} D_1^{-\beta_1 \mathtt{T} - \beta_3} = g_2^{x_0 + s_1(x_1 \mathtt{T} + x_3)},\\
&\mathtt{D}'_{x,2} := D'_{x,2} D_2^{-\beta_1 \mathtt{T} - \beta_3} = g_2^{s_2(x_1 \mathtt{T} + x_3)},\\
&\mathtt{D}''_{x,i} := D''_{x,i} D_i^{-\beta_4} = g_2^{s_i x_4},\\
&\mathtt{D}_{y,i} := D_{y,i} D_i^{\beta'_2} = g_2^{-s_i y_2},\\
&\mathtt{D}'_{y,1} := D'_{y,1} g_2^{\beta'_0} D_1^{\beta'_1 \mathtt{T} + \beta'_3} = g_2^{-y_0 - s_1(y_1 \mathtt{T} + y_3)},\\
&\mathtt{D}'_{y,2} := D'_{y,2} D_2^{\beta'_1 \mathtt{T} + \beta'_3} = g_2^{-s_2(y_1 \mathtt{T} + y_3)},\\
&\mathtt{D}''_{y,i} := D''_{y,i} D_i^{\beta'_4} = g_2^{-s_i y_4}.
\end{aligned}$$

Output

$$\begin{aligned}
\mathsf{mk}_{\mathtt{T}} := (&g_2, \{\beta_i, \beta'_i\}_{i=0}^4,\\
&\{\mathtt{D}_i, \mathtt{D}_{x,i}, \mathtt{D}'_{x,i}, \mathtt{D}''_{x,i}, \mathtt{D}_{y_i}, \mathtt{D}'_{y,i}, \mathtt{D}''_{y,i}\}_{i=1}^2).
\end{aligned}$$

- $\mathsf{KG}(\mathsf{prms}, \mathsf{mk}_{\mathtt{T}}, \mathtt{I})$: Choose $r \leftarrow \mathbb{Z}_p$ and compute

$$\begin{aligned}
\mathtt{K} &:= \mathtt{D}_1 (\mathtt{D}_2)^r = g_2^{s_1 + s_2 r},\\
\mathtt{K}_x &:= \mathtt{D}_{x,1} (\mathtt{D}_{x,2})^r = g_2^{x_2(s_1 + s_2 r)},\\
\mathtt{K}'_x &:= \mathtt{D}'_{x,1} (\mathtt{D}''_{x,1})^{\mathtt{I}} \left(\mathtt{D}'_{x,2} (\mathtt{D}''_{x,1})^{\mathtt{I}}\right)^r\\
&= g_2^{x_0 + (s_1 + s_2 r)(x_1 \mathtt{T} + x_3 + x_4 \mathtt{I})},\\
\mathtt{K}_y &:= \mathtt{D}_{y,1} (\mathtt{D}_{y,2})^r = g_2^{-y_2(s_1 + s_2 r)},\\
\mathtt{K}'_y &:= \mathtt{D}'_{y,1} (\mathtt{D}''_{y,1})^{\mathtt{I}} \left(\mathtt{D}'_{y,2} (\mathtt{D}''_{y,1})^{\mathtt{I}}\right)^r\\
&= g_2^{-y_0 - (s_1 + s_2 r)(y_1 \mathtt{T} + y_3 + y_4 \mathtt{I})},
\end{aligned}$$

Output

$$\mathsf{dk}_{\mathtt{T},\mathtt{I}} := (\mathtt{K}, \mathtt{K}_x, \mathtt{K}'_x, \mathtt{K}_y, \mathtt{K}'_y).$$

- $\mathsf{IBEnc}(\mathsf{prms}, M, \mathtt{T}, \mathtt{I})$: Choose $t, \mathtt{tag} \xleftarrow{\$} \mathbb{Z}_p$. For $M \in \mathbb{G}_T$, compute

$$\begin{aligned}
&C_M := M \cdot z^t, \quad C_x := (g_1^{\alpha})^t,\\
&C_y := g_1^t, \quad C := \left(u_1^{\mathtt{T}} w_1^{\mathtt{tag}} h_1 v_1^{\mathtt{I}}\right)^t,
\end{aligned}$$

Output $\mathsf{ct}_{\mathtt{T},\mathtt{I}} := (C_M, C_x, C_y, C, \mathtt{tag})$.

- $\mathsf{IBDec}(\mathsf{prms}, \mathsf{dk}_{\mathtt{T},\mathtt{I}}, \mathsf{ct}_{\mathtt{T},\mathtt{I}})$: Compute and output

$$M = \frac{C_M \cdot e(C, \mathtt{K})}{e(C_x, \mathtt{K}_x^{\mathtt{tag}} \mathtt{K}'_x) e(C_y, \mathtt{K}_y^{\mathtt{tag}} \mathtt{K}'_y)}.$$

We show the decryption correctness of the above construction. Let $\mathsf{dk}_{\mathtt{T},\mathtt{I}} = (\mathtt{K}, \mathtt{K}_x, \mathtt{K}'_x, \mathtt{K}_y, \mathtt{K}'_y)$ and $\mathsf{ct}_{\mathtt{T},\mathtt{I}} = (C_M, C_x, C_y, C, \mathtt{tag})$ be the correctly-generated decryption key and ciphertext. We then have

$$\begin{aligned}
&e(C, \mathtt{K})\\
&= e\left(\left(u_1^{\mathtt{T}} w_1^{\mathtt{tag}} h_1 v_1^{\mathtt{I}}\right)^t, g_2^s\right)\\
&= e\left(g_1^{\alpha(x_1 \mathtt{T} + x_2 \mathtt{tag} + x_3 + x_4 \mathtt{I}) + y_1 \mathtt{T} + y_2 \mathtt{tag} + y_3 + y_4 \mathtt{I}}, g_2\right)^{ts}\\
&= e\left(g_1^{x_1 \mathtt{T} + x_2 \mathtt{tag} + x_3 + x_4 \mathtt{I}}, g_2\right)^{\alpha t s}\\
&\qquad \cdot e\left(g_1^{y_1 \mathtt{T} + y_2 \mathtt{tag} + y_3 + y_4 \mathtt{I}}, g_2\right)^{ts},\\
&e(C_x, \mathtt{K}_x^{\mathtt{tag}} \mathtt{K}'_x)\\
&= e\left(g_1^{\alpha t}, g_2^{x_2 \mathtt{stag}} \cdot g_2^{x_0 + s(x_1 \mathtt{T} + x_3 + x_4 \mathtt{I})}\right)\\
&= e\left(g_1^{\alpha t}, g_2^{x_0}\right) \cdot e\left(g_1, g_2^{x_1 \mathtt{T} + x_2 \mathtt{tag} + x_3 + x_4 \mathtt{I}}\right)^{\alpha t s},\\
&e(C_y, \mathtt{K}_y^{\mathtt{tag}} \mathtt{K}'_y)\\
&= e\left(g_1^t, g_2^{y_2 \mathtt{stag}} \cdot g_2^{y_0 + s(y_1 \mathtt{T} + y_3 + y_4 \mathtt{I})}\right)
\end{aligned}$$

$$= e\left(g_1^t, g_2^{y_0}\right) \cdot e\left(g_1, g_2^{y_1\mathtt{T}+y_2\mathtt{tag}+y_3+y_4\mathtt{I}}\right)^{ts}.$$

Hence, we have

$$\frac{e(C,\mathtt{K})}{e(C_x, \mathtt{K}_x^{\mathtt{tag}}\mathtt{K}'_x)e(C_y, \mathtt{K}_y^{\mathtt{tag}}\mathtt{K}'_y)} = e\left(g_1, g_2\right)^{-t(x_0\alpha+y_0)},$$

which ensures the decryption correctness.

Our construction is secure under the following symmetric external Diffie-Hellman (SXDH) assumption. In a nutshell, we say that the SXDH assumption holds if the decisional Diffie-Hellman (DDH) assumption holds both in $\mathbb{G}_1$ and in $\mathbb{G}_2$. It can be rephrased as "both the DDH1 and DDH2 assumptions hold".

Formally, those assumptions are defined as follows. Let $\mathcal{A}$ be a PPT adversary, and we consider the following game against $\mathcal{A}$.

$\mathsf{Exp}^{\mathsf{DDH}i}_{\mathcal{G},\mathcal{A}}(1^\lambda)$

$D := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow \mathcal{G}(1^\lambda)$

$c_1, c_2, \mu \xleftarrow{\$} \mathbb{Z}_p, \quad b \xleftarrow{\$} \{0,1\}$

if $b = 0$ **then** $T := g_i^{c_1 c_2}$ **else** $T := g_i^{c_1 c_2 + \mu}$

$b' \leftarrow \mathcal{A}(D, g_i^{c_1}, g_i^{c_2}, T)$

if $b' = b$ **return** 1 **else return** 0

Definition 12 (DDHi assumption) *We say that the* DDHi *assumption relative to a generator $\mathcal{G}$ holds if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{DDH}i}_{\mathcal{G},\mathcal{A}}(1^\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{DDH}i}_{\mathcal{G},\mathcal{A}}(1^\lambda) = 1] - 1/2|$ is negligible in λ.*

Definition 13 (SXDH assumption) *We say that the* SXDH *assumption relative to a generator $\mathcal{G}$ holds if both of the* DDH1 *and* DDH2 *assumptions relative to a generator $\mathcal{G}$ hold.*

We obtain the following theorems, and give the proofs of them in the next subsections.

Theorem 3 *If the* SXDH *assumption relative to $\mathcal{G}$ holds, then the above MIKE scheme $\mathcal{MIKE}$ is* IND-ID-KI-CPA *secure.*

Theorem 4 *If the* SXDH *assumption relative to $\mathcal{G}$ holds, then the above MIKE scheme $\mathcal{MIKE}$ is* ANO-ID-KI-CPA *secure.*

6.1 Semi-functional Ciphertexts and Secret Keys

We prove the both of the IND-ID-KI-CPA security and the ANO-ID-KI-CPA security by employing the dual-system-encryption methodology [31]. Therefore, we first define semi-functional ciphertexts and secret keys as follows.

Semi-functional Ciphertext: Suppose a normal ciphertext $\mathsf{ct}_{\mathtt{T},\mathtt{I}} = (C_M, C_x, C_y, C, \mathtt{tag})$. A semi-functional ciphertext $\widetilde{\mathsf{ct}}_{\mathtt{T},\mathtt{I}} := (\tilde{C}_M, \tilde{C}_x, \tilde{C}_y, \tilde{C}, \widetilde{\mathtt{tag}})$ is computed as follows.

$$\begin{aligned}
\tilde{C}_M &:= C_M e(g_1, g_2)^{x_0\mu} = M e(g_1, g_2)^{x_0(\alpha s+\mu) - y_0 s},\\
\tilde{C}_x &:= C_x g_1^{\mu} = g_1^{\alpha s+\mu},\\
\tilde{C}_y &:= C_y,\\
\tilde{C} &:= C\Big((g_1^{x_1})^{\mathtt{T}}(g_1^{x_2})^{\mathtt{tag}} g_1^{x_3}(g_1^{x_4})^{\mathtt{I}}\Big)^{\mu}\\
&= C_3 g_1^{\mu(x_1\mathtt{T}+x_2\mathtt{tag}+x_3+x_4\mathtt{I})}\\
&= g_1^{(\alpha s+\mu)(x_1\mathtt{T}+x_2\mathtt{tag}+x_3+x_4\mathtt{I})} g_1^{-s(y_1\mathtt{T}+y_2\mathtt{tag}+y_3+y_4\mathtt{I})},
\end{aligned}$$

and $\widetilde{\mathtt{tag}} := \mathtt{tag}$, where $\mu \xleftarrow{\$} \mathbb{Z}_p^\times$.

Partial Semi-functional Master Key: Suppose a normal master key $\mathsf{mk}_{\mathtt{T}} = (g_2, \{\beta_i, \beta'_i\}_{i=0}^4, \mathcal{D}_1, \mathcal{D}_2)$, where $\mathcal{D}_i := \{\mathtt{D}_i, \mathtt{D}_{x,i}, \mathtt{D}'_{x,i}, \mathtt{D}''_{x,i}, \mathtt{D}_{y,i}, \mathtt{D}'_{y,i}, \mathtt{D}''_{y,i}\}$. A partial semi-functional master key $\widehat{\mathsf{mk}}_{\mathtt{T}} := (g_2, \{\beta_i, \beta'_i\}_{i=0}^4, \widehat{\mathcal{D}}_1, \mathcal{D}_2)$ for $\mathtt{T}$, where $\widehat{\mathcal{D}}_1 := \{\hat{\mathtt{D}}_1, \hat{\mathtt{D}}_{x,1}, \hat{\mathtt{D}}'_{x,1}, \hat{\mathtt{D}}''_{x,1}, \hat{\mathtt{D}}_{y,1}, \hat{\mathtt{D}}'_{y,1}, \hat{\mathtt{D}}''_{y,1}\}$, is computed as follows.

$$\begin{aligned}
\hat{\mathtt{D}}_1 &:= \mathtt{D}_1,\\
\hat{\mathtt{D}}_{x,1} &= \mathtt{D}_{x,1} g_2^{\frac{\gamma_1}{\alpha}} = g_2^{s_1 x_2 + \frac{\gamma_1}{\alpha}},\\
\hat{\mathtt{D}}'_{x,1} &:= \mathtt{D}'_{x,1} g_2^{\frac{\gamma_1\phi_1}{\alpha}} = g_2^{x_0 + s_1(x_1\mathtt{T}+x_3) + \frac{\gamma_1\phi_1}{\alpha}},\\
\hat{\mathtt{D}}''_{x,1} &= \mathtt{D}''_{x,1} g_2^{\frac{\gamma_1\psi_1}{\alpha}} = g_2^{s_1 x_4 + \frac{\gamma_1\psi_1}{\alpha}},\\
\hat{\mathtt{D}}_{y,1} &= \mathtt{D}_{y,1} g_2^{-\gamma_1} = g_2^{-s_1 y_2 - \gamma_1},\\
\hat{\mathtt{D}}'_{y,1} &:= \mathtt{D}'_{y,1} g_2^{-\gamma_1\phi_1} = g_2^{-y_0 - s_1(y_1\mathtt{T}+y_3) - \gamma_1\phi_1},\\
\hat{\mathtt{D}}''_{y,1} &= \mathtt{D}''_{y,1} g_2^{-\gamma_1\psi_1} = g_2^{-s_1 y_4 - \gamma_1\psi_1},
\end{aligned}$$

where $\phi_1, \psi_1 \xleftarrow{\$} \mathbb{Z}_p$ and $\gamma_1 \xleftarrow{\$} \mathbb{Z}_p^\times$.

Semi-functional Master Key: Suppose a normal master key $\mathsf{mk}_{\mathtt{T}} = (g_2, \{\beta_i, \beta'_i\}_{i=0}^4, \mathcal{D}_1, \mathcal{D}_2)$, where $\mathcal{D}_i := \{\mathtt{D}_i, \mathtt{D}_{x,i}, \mathtt{D}'_{x,i}, \mathtt{D}''_{x,i}, \mathtt{D}_{y,i}, \mathtt{D}'_{y,i}, \mathtt{D}''_{y,i}\}$. A semi-functional master key $\widehat{\mathsf{mk}}_{\mathtt{T}} := (g_2, \{\beta_i, \beta'_i\}_{i=0}^4, \widehat{\mathcal{D}}_1, \widehat{\mathcal{D}}_2)$ for $\mathtt{T}$, where $\widehat{\mathcal{D}}_i := \{\hat{\mathtt{D}}_i, \hat{\mathtt{D}}_{x,i}, \hat{\mathtt{D}}'_{x,i}, \hat{\mathtt{D}}''_{x,i}, \hat{\mathtt{D}}_{y,i}, \hat{\mathtt{D}}'_{y,i}, \hat{\mathtt{D}}''_{y,i}\}$, is computed as follows.

$$\begin{aligned}
\tilde{\mathtt{D}}_i, &:= \mathtt{D}_i \quad (\text{for } i \in \{1,2\}),\\
\tilde{\mathtt{D}}_{x,i} &= \mathtt{D}_{x,i} g_2^{\frac{\gamma_i}{\alpha}} = g_2^{s_i x_2 + \frac{\gamma_i}{\alpha}} \quad (\text{for } i \in \{1,2\}),\\
\tilde{\mathtt{D}}'_{x,1} &:= \mathtt{D}'_{x,1} g_2^{\frac{\gamma_1\phi_1}{\alpha}} = g_2^{x_0 + s_1(x_1\mathtt{T}+x_3) + \frac{\gamma_1\phi_1}{\alpha}},\\
\tilde{\mathtt{D}}'_{x,2} &:= \mathtt{D}'_{x,2} g_2^{\frac{\gamma_2\phi_2}{\alpha}} = g_2^{s_2(x_1\mathtt{T}+x_3) + \frac{\gamma_2\phi_2}{\alpha}},\\
\tilde{\mathtt{D}}''_{x,i} &= \mathtt{D}''_{x,i} g_2^{\frac{\gamma_i\psi_i}{\alpha}} = g_2^{s_i x_4 + \frac{\gamma_i\psi_i}{\alpha}} \quad (\text{for } i \in \{1,2\}),\\
\tilde{\mathtt{D}}_{y,i} &= \mathtt{D}_{y,i} g_2^{-\gamma_i} = g_2^{-s_i y_2 - \gamma_i} \quad (\text{for } i \in \{1,2\}),\\
\tilde{\mathtt{D}}'_{y,1} &:= \mathtt{D}'_{y,1} g_2^{-\gamma_1\phi_1} = g_2^{-y_0 - s_1(y_1\mathtt{T}+y_3) - \gamma_1\phi_1},\\
\tilde{\mathtt{D}}'_{y,2} &:= \mathtt{D}'_{y,2} g_2^{-\gamma_2\phi_2} = g_2^{-s_2(y_1\mathtt{T}+y_3) - \gamma_2\phi_2},
\end{aligned}$$

$\tilde{\mathtt{D}}''_{y,i} = \mathtt{D}''_{y,i} g_2^{-\gamma_i \psi_i} = g_2^{-s_i y_4 - \gamma_i \psi_i}$ (for $i \in \{1,2\}$),

where $\phi_1, \phi_2, \psi_1, \psi_2 \xleftarrow{\$} \mathbb{Z}_p$ and $\gamma_1, \gamma_2 \xleftarrow{\$} \mathbb{Z}_p^\times$.

Semi-functional Decryption Key: Suppose a normal decryption key $\mathsf{dk}_{\mathtt{T},\mathtt{I}} = (\mathtt{K}, \mathtt{K}_x, \mathtt{K}'_x, \mathtt{K}_y, \mathtt{K}'_y)$. A semi-functional decryption key $\widetilde{\mathsf{dk}}_{\mathtt{T},\mathtt{I}} := (\tilde{\mathtt{K}}, \tilde{\mathtt{K}}_x, \tilde{\mathtt{K}}'_x, \tilde{\mathtt{K}}_y, \tilde{\mathtt{K}}'_y)$ for $\mathtt{T}$ is computed as follows.

$\tilde{\mathtt{K}}, := \mathtt{K},$
$\tilde{\mathtt{K}}_x = \mathtt{K}_x g_2^{\frac{\gamma}{\alpha}} = g_2^{s x_2 + \frac{\gamma}{\alpha}},$
$\tilde{\mathtt{K}}'_x := \mathtt{K}'_x g_2^{\frac{\gamma\phi}{\alpha}} = g_2^{x_0 + s(x_1 \mathtt{T} + x_3) + \frac{\gamma\phi}{\alpha}},$
$\tilde{\mathtt{K}}_y = \mathtt{K}_y g_2^{-\gamma} = g_2^{-s y_2 - \gamma},$
$\tilde{\mathtt{K}}'_y := \mathtt{K}'_y g_2^{-\gamma\phi} = g_2^{-y_0 - s(y_1 \mathtt{T} + y_3) - \gamma\phi},$

where $\phi \xleftarrow{\$} \mathbb{Z}_p$ and $\gamma \xleftarrow{\$} \mathbb{Z}_p^\times$. Note that s is internal randomness of $\mathsf{dk}_{\mathtt{T},\mathtt{I}}$.

Note that decryption keys generated from a (partial) semi-functional master key is also semi-functional. We also note that in order to generate the above keys, $g_2^{\frac{1}{\alpha}}$ is needed in addition to the public parameter. A semi-functional ciphertext for $\mathtt{I}$ for $\mathtt{T}$ can be decrypted with a decryption key for $\mathtt{I}$ and $\mathtt{T}$. This fact can be easily checked by

$$\frac{e(g_1, g_2)^{x_0 \mu} e(g_1^{\mu(x_1 \mathtt{T} + x_2 \mathtt{tag} + x_3 + x_4 \mathtt{I})}, \mathtt{K})}{e(g_1^{\mu}, \mathtt{K}_x^{\mathtt{tag}} \mathtt{K}'_x)} = 1_{\mathbb{G}_T},$$

where $1_{\mathbb{G}_T}$ is the identity element of $\mathbb{G}_T$. Also, a normal ciphertext can be decrypted with a semi-functional decryption key since it holds

$$e(C_x, g_2^{\frac{\gamma}{\alpha}\mathtt{tag}} g_2^{\frac{\gamma\phi}{\alpha}}) e(C_y, g_2^{-\gamma \mathtt{tag}} g_2^{-\gamma\phi}) = 1_{\mathbb{G}_T}.$$

6.2 Proof of Theorem 3

We prove the theorem through a sequence of games. We first define the following games:

$\mathsf{Game}_{\mathsf{Real}}$: This is the same as the IND-ID-KI-CPA game.

Game_0: This is the same as $\mathsf{Game}_{\mathsf{Real}}$ except that the challenge ciphertext is semi-functional.

$\mathsf{Game}_{1,k}$ ($1 \le k \le q_T$): This is the same as Game_0 except for the following modification: Let q_T be the maximum number of time-periods issued to the oracle $\mathcal{O}_{\mathrm{LEAK}}$, and $\mathtt{T}^{(i)}$ ($1 \le i \le q_T$) be an i-th time-period issued to the oracle. If queries regarding the first k time-periods $\mathtt{T}^{(1)}, \ldots, \mathtt{T}^{(k)}$ are issued, then partial semi-functional master keys are returned. The rest of keys (i.e., keys regarding $\mathtt{T}^{(k+1)}, \ldots, \mathtt{T}^{(q_T)}$) are normal. Note that all the decryption keys returned from the oracle $\mathcal{O}_{\mathrm{EXT}}$ are normal.

$\mathsf{Game}_{2,k}$ ($1 \le k \le q_T$): This is the same as Game_{1,q_T} except for the following modification: If queries regarding the first k time-periods $\mathtt{T}^{(1)}, \ldots, \mathtt{T}^{(k)}$ are issued to the oracle $\mathcal{O}_{\mathrm{LEAK}}$, then semi-functional master keys are returned. The rest of master keys are partial semi-functional. Note that all the decryption keys returned from the oracle $\mathcal{O}_{\mathrm{EXT}}$ are still normal.

$\mathsf{Game}_{3,k}$ ($1 \le k \le q_I$): This is the same as Game_{2,q_T} except for the following modification: Let q_I be the maximum number of queries issued to the oracle $\mathcal{O}_{\mathrm{EXT}}$, and $(\mathtt{T}^{(i_j)}, \mathtt{I}^{(i_j)})$ ($1 \le j \le q_I$) be an j-th query issued to the oracle. If queries regarding the first k queries $(\mathtt{T}^{(i_1)}, \mathtt{I}^{(i_1)}), (\mathtt{T}^{(i_2)}, \mathtt{I}^{(i_2)}), \ldots, (\mathtt{T}^{(i_k)}, \mathtt{I}^{(i_k)})$ are issued to the oracle $\mathcal{O}_{\mathrm{EXT}}$, then semi-functional decryption keys are returned. The rest of decryption keys are normal.

$\mathsf{Game}_{\mathsf{Final}}$: This is the same as $\mathsf{Game}_{2,q}$ except that the challenge ciphertext is a semi-functional one of a random element of $\mathbb{G}_T$.

Let S_{Real}, $S_{1,k}$ ($0 \le k \le q_T$), $S_{2,k}$ ($0 \le k \le q_T$), $S_{3,k}$ ($0 \le k \le q_I$), and S_{Final} be the probabilities that the event $b' = b$ occurs in $\mathsf{Game}_{\mathsf{Real}}$, $\mathsf{Game}_{1,k}$, $\mathsf{Game}_{2,k}$, $\mathsf{Game}_{3,k}$, and $\mathsf{Game}_{\mathsf{Final}}$, respectively. Since $|S_{\mathsf{Final}} - 1/2| = 0$, we have

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{ID\text{-}KI\text{-}CPA}}_{\mathcal{MIKE},\mathcal{A}}(1^\lambda) \\
&\le |S_{\mathsf{Real}} - S_0| + \sum_{i=1}^{q_T} |S_{1,i-1} - S_{1,i}| + \sum_{i=1}^{q_T} |S_{2,i-1} - S_{2,i}| \\
&\qquad + \sum_{i=1}^{q_I} |S_{3,i-1} - S_{3,i}| + |S_{3,q} - S_{\mathsf{Final}}|,
\end{aligned}$$

where $S_{1,0} := S_0$, $S_{2,0} := S_{1,q}$, and $S_{3,0} := S_{2,q}$. The rest of the proof follows from the following lemmas.

Lemma 3 $|S_{\mathsf{Real}} - S_0| \le 2\mathsf{Adv}^{\mathsf{DDH1}}_{\mathcal{G},\mathcal{B}}(1^\lambda)$.

Proof At the beginning, a PPT adversary $\mathcal{B}$ receives an instance $(g_1, g_1^{c_1}, g_1^{c_2}, g_2, T)$ of the DDH1 problem. Then, $\mathcal{B}$ randomly chooses $x_i, y_i \xleftarrow{\$} \mathbb{Z}_p$ with $i \in \{0, 1, \ldots, 4\}$, and creates

$z := e(g_1^{c_1}, g_2)^{x_0} e(g_1, g_2)^{-y_0}, \quad u_1 := (g_1^{c_1})^{x_1} g_1^{-y_1},$
$w_1 := (g_1^{c_1})^{x_2} g_1^{-y_2}, \quad h_1 := (g_1^{c_1})^{x_3} g_1^{-y_3},$
$v_1 := (g_1^{c_1})^{x_4} g_1^{-y_4}.$

$\mathcal{B}$ sends $\mathsf{prms} := (g_1, g_1^{\alpha}, u_1, w_1, h_1, v_1, z)$ to $\mathcal{A}$. $\mathcal{B}$ also chooses $\beta_i, \beta'_i, \leftarrow \mathbb{Z}_p$ with $i \in \{0, 1, \ldots, 4\}$. Note that $\mathcal{B}$ knows a master helper key mhk and an initial master key mk_0, and we implicitly set $\alpha := c_1$.

$\mathcal{B}$ can simulate all the oracles since $\mathcal{B}$ knows mhk and mk_0.

In the challenge phase, $\mathcal{B}$ receives $(M_0^*, M_1^*, \mathtt{I}^*, \mathtt{T}^*)$ from $\mathcal{A}$. $\mathcal{B}$ chooses $d \xleftarrow{\$} \{0,1\}$. $\mathcal{B}$ chooses $\mathtt{tag}^* \xleftarrow{\$} \mathbb{Z}_p$ and computes

$$C_M^* := M_d^* e(T, g_2)^{x_0} e(g_1^{c_2}, g_2)^{-y_0}, \quad C_x^* := T, \quad C_y^* := g_1^{c_2},$$
$$C^* := T^{x_1 \mathtt{T}^* + x_2 \mathtt{tag}^* + x_3 + x_4 \mathtt{I}^*} (g_1^{c_2})^{-y_1 \mathtt{T}^* - y_2 \mathtt{tag}^* - y_3 - y_4 \mathtt{I}^*}.$$

$\mathcal{B}$ sends $C^*_{\mathtt{T}^*, \mathtt{I}^*} := (C_M^*, C_x^*, C_y^*, C^*, \mathtt{tag}^*)$ to $\mathcal{A}$.

If $b = 0$, then the above ciphertext is normal by setting $t := c_2$. If $b = 1$, then the above ciphertext is semi-functional since it holds

$$\begin{aligned}
C_M^* &= M_d^* e(g_1, g_2)^{x_0(c_1 c_2 + \mu) - y_0 c_2} \\
&= M_d^* e(g_1, g_2)^{x_0(\alpha s + \mu) - y_0 s}, \\
C_x^* &= g_1^{c_1 c_2 + \mu} = g_1^{\alpha s + \mu}, \\
C^* &= g^{(c_1 c_2 + \mu)(x_1 \mathtt{T}^* + x_2 \mathtt{tag}^* + x_3 + x_4 \mathtt{I}^*)} \\
&\qquad \cdot g_1^{-c_2(y_1 \mathtt{T}^* + y_2 \mathtt{tag}^* + y_3 + y_4 \mathtt{I}^*)} \\
&= g^{(\alpha s + \mu)(x_1 \mathtt{T}^* + x_2 \mathtt{tag}^* + x_3 + x_4 \mathtt{I}^*)} \\
&\qquad \cdot g_1^{-s(y_1 \mathtt{T}^* + y_2 \mathtt{tag}^* + y_3 + y_4 \mathtt{I}^*)}.
\end{aligned}$$

After receiving d' from $\mathcal{A}$, $\mathcal{B}$ sends $b' = 1$ to the challenger of the DDH1 problem if $d' = d$. Otherwise, $\mathcal{B}$ sends $b' = 0$ to the challenger. □

Lemma 4 *For every $k \in \{1, 2, \ldots, q_T\}$, $|S_{1,k-1} - S_{1,k}| \leq 2\mathsf{Adv}^{\mathsf{DDH2}}_{\mathcal{G},\mathcal{B}}(1^\lambda)$.*

Proof At the beginning, a PPT adversary $\mathcal{B}$ receives an instance $(g_1, g_2, g_2^{c_1}, g_2^{c_2}, T)$ of the DDH2 problem. Then, $\mathcal{B}$ randomly chooses $x_0, x_1', x_2', x_3', x_4', y_0, y_1', y_1'', y_3', y_3'', y_4', y_4'' \xleftarrow{\$} \mathbb{Z}_p$ and $\alpha \xleftarrow{\$} \mathbb{Z}_p^\times$, and (implicitly) sets

$$x_1 := \frac{x_1' + y_1}{\alpha}, \text{ where } y_1 := y_1' + c_2 y_1'',$$
$$x_2 := \frac{x_2' + c_2}{\alpha}, \quad y_2 := c_2,$$
$$x_3 := \frac{x_3' + y_3}{\alpha}, \text{ where } y_3 := y_3' + c_2 y_3'',$$
$$x_4 := \frac{x_4' + y_4}{\alpha}, \text{ where } y_4 := y_4' + c_2 y_4''.$$

$\mathcal{B}$ creates

$$z := e(g_1, g_2)^{x_0 \alpha - y_0}, \quad u_1 := g_1^{x_1'}, \quad w_1 := g_1^{x_2'},$$
$$h_1 := g_1^{x_3'}, \quad v_1 := g_1^{x_4'}.$$

$\mathcal{B}$ sends $\mathsf{prms} := (g_1, g_1^\alpha, u_1, w_1, h_1, v_1, z)$ to $\mathcal{A}$. $\mathcal{B}$ also chooses $\beta_i, \beta_i' \leftarrow \mathbb{Z}_p$ with $i \in \{0, 1, \ldots, 4\}$. Note that $\mathcal{B}$ does not know a master helper key mhk but knows an initial master key mk_0.

$\mathcal{B}$ knows x_0 and y_0, and can compute

$$g_2^{x_i} := g_2^{\frac{x_i' + y_i'}{\alpha}} (g_2^{c_2})^{\frac{y_i''}{\alpha}} \text{ for } i \in \{1,3,4\}, \quad g_2^{x_2} := g_2^{\frac{x_2'}{\alpha}} (g_2^{c_2})^{\frac{1}{\alpha}},$$
$$g_2^{y_i} := g_2^{y_i'} (g_2^{c_2})^{y_i''} \text{ for } i \in \{1,3,4\}, \quad g_2^{x_2} := g_2^{c_2}.$$

Therefore, $\mathcal{B}$ can simulate the oracle $\mathcal{O}_{\mathrm{EXT}}$.

We show how $\mathcal{B}$ simulates the oracle $\mathcal{O}_{\mathrm{LEAK}}$ for a query $\mathtt{T} \in \mathcal{T}$ (not $\star$). Let $\mathtt{T}^{(i)}$ be an i-th time-period issued to the oracle. $\mathcal{B}$ creates $k-1$ partial semi-functional master keys, and embeds T into keys for the k-th period. The rest of keys are normal.

Case $i < k$ and Case $i > k$: After receiving $\mathtt{T}^{(i)}$, $\mathcal{B}$ creates and returns semi-functional decryption keys. Since $\mathcal{B}$ knows $(g_2^{x_i}, g_2^{y_i})$ $(i \in \{0, 1, \ldots, 4\})$, $\mathcal{B}$ can create both normal and partial semi-functional master keys.

Case $i = k$: After receiving $\mathtt{T}^{(k)}$, $\mathcal{B}$ creates a master key for $\mathtt{T}^{(k)}$ by embedding T as follows. $\mathcal{B}$ computes

$$\begin{aligned}
\mathtt{D}_1 &:= g_2^{c_1}, \\
\mathtt{D}_{x,1} &:= \left((g_2^{c_1})^{x_2'} T\right)^{\frac{1}{\alpha}}, \\
\mathtt{D}'_{x,1} &:= g_2^{x_0} (g_2^{c_1})^{\frac{\mathtt{T}^{(k)}(x_1' + y_1') + x_3' + y_3'}{\alpha}} T^{\frac{\mathtt{T}^{(k)} y_1'' + y_3''}{\alpha}}, \\
\mathtt{D}''_{x,1} &:= \left((g_2^{c_1})^{x_4' + y_4'} T^{y_4''}\right)^{\frac{1}{\alpha}}, \\
\mathtt{D}_{y,1} &:= T^{-1}, \\
\mathtt{D}'_{y,1} &:= g_2^{-y_0} (g_2^{c_1})^{-\mathtt{T}^{(k)} y_1' - y_3'} T^{-\mathtt{T}^{(k)} y_1'' - y_3''}, \\
\mathtt{D}''_{y,1} &:= (g_2^{c_1})^{-y_4'} T^{-y_4''}.
\end{aligned}$$

$\mathcal{B}$ sets $\mathsf{mk}_{\mathtt{T}^{(k)}} := (g_2, \{\beta_i, \beta_i'\}_{i=0}^4, \{\mathtt{D}_i, \mathtt{D}_{x,i}, \mathtt{D}'_{x,i}, \mathtt{D}''_{x,i}, \mathtt{D}_{y,i}, \mathtt{D}'_{y,i}, \mathtt{D}''_{y,i}\}_{i=1}^2)$. If $b = 0$, then it is easy to see that the above key is normal by setting $s_1 := c_1$. If $b = 1$, then the above master key is partial semi-functional since it holds

$$\begin{aligned}
\mathtt{D}_x &:= \left((g_2^{c_1})^{x_2'} T\right)^{\frac{1}{\alpha}} = g_2^{\frac{c_1(x_2' + c_2)}{\alpha}} g_2^{\frac{\gamma_1}{\alpha}} = g_2^{s_1 x_2} g_2^{\frac{\gamma_1}{\alpha}}, \\
\mathtt{D}'_x &:= g_2^{x_0} (g_2^{c_1})^{\frac{\mathtt{T}^{(k)}(x_1' + y_1') + x_3' + y_3'}{\alpha}} T^{\frac{\mathtt{T}^{(k)} y_1'' + y_3''}{\alpha}} \\
&= g_2^{x_0 + c_1 \cdot \frac{\mathtt{T}^{(k)}(x_1' + y_1' + y_1'' c_2) + x_3' + y_3' + y_3'' c_2}{\alpha}} g_2^{\frac{\gamma_1 (\mathtt{T}^{(k)} y_1'' + y_3'')}{\alpha}} \\
&= g_2^{x_0 + s_1(\mathtt{T}^{(k)} x_1 + x_3)} g_2^{\frac{\gamma_1 \phi_1}{\alpha}} \\
\mathtt{D}''_{x,1} &:= \left((g_2^{c_1})^{x_4' + y_4'} T^{y_4''}\right)^{\frac{1}{\alpha}} \\
&= g_2^{\frac{c_1(x_4' + y_4' + c_2 y_4'')}{\alpha}} g_2^{\frac{\gamma_1 y_4''}{\alpha}} = g_2^{s_1 x_4} g_2^{\frac{\gamma_1 \psi_1}{\alpha}}, \\
\mathtt{D}_y &:= T^{-1} = g_2^{-c_1 c_2 - \gamma_1} = g_2^{-s_1 y_2 - \gamma_1}, \\
\mathtt{D}'_y &:= g_2^{-y_0} (g_2^{c_1})^{-\mathtt{T}^{(k)} y_1' - y_3'} T^{-\mathtt{T}^{(k)} y_1'' - y_3''} \\
&= g_2^{-y_0 - c_1(\mathtt{T}^{(k)}(y_1' + y_1'' c_2) + y_3' + y_3'' c_2)} g_2^{-\gamma_1 (\mathtt{T}^{(k)} y_1'' + y_3'')} \\
&= g_2^{-y_0 - s_1(\mathtt{T}^{(k)} y_1 + y_3 + \mathtt{I} y_4)} g_2^{-\gamma_1 \phi_1}, \\
\mathtt{D}''_{y,1} &:= (g_2^{c_1})^{-y_4'} T^{-y_4''} \\
&= g_2^{-c_1(y_4' + c_2 y_4'')} g_2^{-\gamma_1 y_4''} = g_2^{-s_1 y_4} g_2^{-\gamma_1 \psi_1},
\end{aligned}$$

where $T := g_2^{c_1 c_2 + \gamma_1}$, $s_1 := c_1$, $\phi_1 := \mathtt{T}^{(k)} y_1'' + y_3''$, and $\psi_1 := y_4''$. Since y_1'' and y_3'' are chosen uniformly at random, ϕ_1 is uniformly distributed.

We next show how $\mathcal{B}$ simulate the $\mathcal{O}_{\mathrm{LEAK}}$ oracle for a query $\star$. Note that in this case, $\mathcal{A}$ cannot issue any other queries to the oracle $\mathcal{O}_{\mathrm{LEAK}}$. To create mhk, $\mathcal{B}$ randomly chooses $\zeta_i, \zeta_i' \xleftarrow{\$} \mathbb{Z}_p$ with $i \in \{1, \ldots, 4\}$, and implicitly sets "new" noises β_i and β_i' as follows.

$$\beta_i := \zeta_i - x_i, \quad \beta_i' := -\zeta_i' - y_i,$$

where $1 \le i \le 4$. Then, $\mathcal{B}$ computes $B_{x,0} := g_2^{x_0 + \beta_0}$ and $B_{y,0} := g_2^{-y_0 - \beta_0'}$, and sets $B_{x,i} := g_2^{\zeta_i}$ and $B_{y,i} := g_2^{\zeta_i'}$ with $i \in \{1, \ldots, 4\}$. Although $\mathcal{B}$ does not know the values of $\{\beta_i, \beta_i'\}_{i=1}^4$, the proof works well since $\mathcal{A}$ never issues any queries to the oracle $\mathcal{O}_{\mathrm{LEAK}}$.

In the challenge phase, $\mathcal{B}$ receives $(M_0^*, M_1^*, \mathtt{T}^*, \mathtt{I}^*)$ from $\mathcal{A}$. $\mathcal{B}$ chooses $d \xleftarrow{\$} \{0,1\}$. However, $\mathcal{B}$ cannot create the semi-functional ciphertext for $\mathtt{I}^*$ for $\mathtt{T}^*$ without knowledge of c_2 (and hence y_1, y_3, and y_4). To generate the semi-functional ciphertext without the knowledge, $\mathcal{B}$ sets

$$\widetilde{\mathtt{tag}}^* := \mathtt{T}^* y_1'' - y_3'' - \mathtt{I}^* y_4''.$$

Since y_1'', y_3'', and y_4'' are chosen uniformly at random, probability distribution of $\widetilde{\mathtt{tag}}^*$ is also uniformly at random from $\mathcal{A}$'s view. Furthermore, $\widetilde{\mathtt{tag}}^*$ is independent of the semi-functional randomness $\phi_1 = \mathtt{T}^{(k)} y_1'' + y_3''$ even if $\mathcal{A}$ computes a decryption key $\widetilde{\mathsf{dk}}_{\mathtt{T}^{(k)},\mathtt{I}}$ for any $\mathtt{I}$ from $\widehat{\mathsf{mk}}_{\mathtt{T}^{(k)}}$ and gets semi-functional randomness $\phi_1 \mathtt{I} \psi_1 = \mathtt{T}^{(k)} y_1'' + y_3'' + \mathtt{I} y_4''$. Namely, $\mathcal{A}$ can get $\widetilde{\mathsf{dk}}_{\mathtt{T}^{(k)},\mathtt{I}}$ for any $\mathtt{I}$ (without re-randomization), where

$$\begin{aligned}
&\tilde{\mathtt{K}} := \hat{\mathtt{D}}_1, \quad \tilde{\mathtt{K}}_x := \hat{\mathtt{D}}_{x,1}, \\
&\tilde{\mathtt{K}}_x' := \hat{\mathtt{D}}_{x,1}' (\hat{\mathtt{D}}_{x,1}'')^{\mathtt{I}} = g_2^{x_0 + s_1(x_1 \mathtt{T} + x_3 + x_4 \mathtt{I})} g_2^{\frac{\phi_1 \mathtt{I} \psi_1}{\alpha}}, \\
&\tilde{\mathtt{K}}_y := \hat{\mathtt{D}}_{y,1}, \quad \tilde{\mathtt{K}}_y' := \hat{\mathtt{D}}_{y,1}' (\hat{\mathtt{D}}_{y,1}'')^{\mathtt{I}},
\end{aligned}$$

from $\widehat{\mathsf{mk}}_{\mathtt{T}^{(k)}} = (g_2, \{\beta_i, \beta_i'\}_{i=0}^4, \{\hat{\mathtt{D}}_i, \hat{\mathtt{D}}_{x,i}, \hat{\mathtt{D}}_{x,i}', \hat{\mathtt{D}}_{x,i}'', \hat{\mathtt{D}}_{y,i}, \hat{\mathtt{D}}_{y,i}', \hat{\mathtt{D}}_{y,i}''\}_{i=1}^2)$. $\widetilde{\mathtt{tag}}^*$ and $\phi_1 \mathtt{I} \psi_1$ are independent of each other from $\mathcal{A}$'s view since we have

$$\begin{pmatrix} \widetilde{\mathtt{tag}}^* \\ \phi_1 \mathtt{I} \psi_1 \end{pmatrix} = \begin{pmatrix} \mathtt{T}^*, & 1, & \mathtt{I}^* \\ \mathtt{T}^{(k)}, & 1, & \mathtt{I} \end{pmatrix} \begin{pmatrix} y_1'' \\ y_3'' \\ y_4'' \end{pmatrix},$$

and it holds $(\mathtt{T}^*, \mathtt{I}^*) \neq (\mathtt{T}^{(k)}, \mathtt{I})$. Then, $\mathcal{B}$ chooses $s \xleftarrow{\$} \mathbb{Z}_p$ and $\mu \xleftarrow{\$} \mathbb{Z}_p^\times$, and computes

$$\begin{aligned}
\tilde{C}_M^* &:= M_d^* z^s e(g_1, g_2)^{x_0 \mu} = M_d^* e(g_1, g_2)^{x_0(\alpha s + \mu) + y_0 s}, \\
\tilde{C}_x^* &:= g_1^{\alpha s + \mu} \\
\tilde{C}_y^* &:= g_1^s,
\end{aligned}$$

$$\begin{aligned}
\tilde{C}^* &:= \left(u_{1,}^{\mathtt{T}^*} w_1^{\widetilde{\mathtt{tag}}^*} h_1 v_1^{\mathtt{I}^*} \right)^s \\
&\qquad \cdot g_1^{\frac{\mu}{\alpha}(\mathtt{T}^*(x_1' + y_1') + x_2' \widetilde{\mathtt{tag}}^* + x_3' + y_3' + \mathtt{I}^*(x_4' + y_4'))} \\
&= \left(u_{1,}^{\mathtt{T}^*} w_1^{\widetilde{\mathtt{tag}}^*} h_1 v_1^{\mathtt{I}^*} \right)^s \cdot g_1^{\mu(\mathtt{T}^* x_1 + \widetilde{\mathtt{tag}}^* x_2 + x_3 + \mathtt{I}^* x_4)} \\
&\qquad \cdot g_1^{\frac{-c_2 \mu}{\alpha} \mu(\mathtt{T}^* y_1'' + \widetilde{\mathtt{tag}}^* + y_3'' + \mathtt{I}^* y_4'')} \\
&= \left(u_{1,}^{\mathtt{T}^*} w_1^{\widetilde{\mathtt{tag}}^*} h_1 v_1^{\mathtt{I}^*} \right)^s \cdot g_1^{\mu(\mathtt{T}^* x_1 + \widetilde{\mathtt{tag}}^* x_2 + x_3 + \mathtt{I}^* x_4)}.
\end{aligned}$$

$\mathcal{B}$ sends $\widetilde{\mathsf{ct}}_{\mathtt{T}^*,\mathtt{I}^*}^* := (\tilde{C}_M^*, \tilde{C}_x^*, \tilde{C}_y^*, \tilde{C}^*, \widetilde{\mathtt{tag}}^*)$ to $\mathcal{A}$.

After receiving d' from $\mathcal{A}$, $\mathcal{B}$ sends $b' = 1$ to the challenger of the $\mathsf{DDH2}$ problem if $d' = d$. Otherwise, $\mathcal{B}$ sends $b' = 0$ to the challenger. □

Lemma 5 *For every $k \in \{1, 2, \ldots, q_T\}$, $|S_{2,k-1} - S_{2,k}| \le 2\mathsf{Adv}_{\mathcal{G},\mathcal{B}}^{\mathsf{DDH2}}(1^\lambda)$.*

Proof (Sketch) We can prove this Theorem in a similar way to Lemma 4, therefore we here give only a proof sketch. As for the k-th query $\mathtt{T}^{(k)}$ to the oracle $\mathcal{O}_{\mathrm{LEAK}}$, $\mathcal{B}$ embeds T into $\mathtt{D}_2, \mathtt{D}_{x,2}, \mathtt{D}_{x,2}', \mathtt{D}_{x,2}'', \mathtt{D}_{y,2}, \mathtt{D}_{y,2}', \mathtt{D}_{y,2}''$ in a way similar to the proof of Lemma 4. If $b = 0$ then $\mathsf{mk}_{\mathtt{T}^{(k)}}$ is partial semi-functional. Otherwise, $\mathsf{mk}_{\mathtt{T}^{(k)}}$ is semi-functional. □

Lemma 6 *For every $k \in \{1, 2, \ldots, q_I\}$, $|S_{3,k-1} - S_{3,k}| \le 2\mathsf{Adv}_{\mathcal{G},\mathcal{B}}^{\mathsf{DDH2}}(1^\lambda)$.*

Proof (Sketch) We can prove this Theorem in a similar way to Lemma 4, therefore we here give only a proof sketch. As for the k-th query $(\mathtt{T}^{(i_k)}, \mathtt{I}^{(i_k)})$ to the oracle $\mathcal{O}_{\mathrm{EXT}}$, $\mathcal{B}$ embeds T in a way similar to the proof of Lemma 4. Specifically, after receiving $(\mathtt{T}^{(i_k)}, \mathtt{I}^{(i_k)})$, $\mathcal{B}$ creates a decryption key for $\mathtt{I}^{(i_k)}$ for $\mathtt{T}^{(i_k)}$ by embedding T as follows. $\mathcal{B}$ computes

$$\begin{aligned}
&\mathtt{K} := g_2^{c_1}, \\
&\mathtt{K}_x := \left((g_2^{c_1})^{x_2'} T \right)^{\frac{1}{\alpha}}, \\
&\mathtt{K}_x' := g_2^{x_0} (g_2^{c_1})^{\frac{\mathtt{T}^{(i_k)}(x_1' + y_1') + x_3' + y_3' + \mathtt{I}^{(i_k)}(x_4' + y_4')}{\alpha}} \\
&\qquad \cdot T^{\frac{\mathtt{T}^{(i_k)} y_1'' + y_3'' + \mathtt{I}^{(i_k)} y_4''}{\alpha}}, \\
&\mathtt{K}_y := T^{-1}, \\
&\mathtt{K}_y' := g_2^{-y_0} (g_2^{c_1})^{-\mathtt{T}^{(i_k)} y_1' - y_3' - \mathtt{I}^{(i_k)} y_4'} \cdot T^{-\mathtt{T}^{(i_k)} y_1'' - y_3'' - \mathtt{I}^{(i_k)} y_4''}.
\end{aligned}$$

$\mathcal{B}$ sets $\mathsf{dk}_{\mathtt{T}^{(i_k)},\mathtt{I}^{(i_k)}} := (\mathtt{K}, \mathtt{K}_x, \mathtt{K}_x', \mathtt{K}_y, \mathtt{K}_y')$. If $b = 0$, then it is easy to see that the above key is normal by setting $s := c_1$. If $b = 1$, then the above decryption key is semi-functional since it holds

$$\begin{aligned}
\mathtt{K}_x &:= \left((g_2^{c_1})^{x_2'} T \right)^{\frac{1}{\alpha}} = g_2^{\frac{c_1(x_2' + c_2)}{\alpha}} g_2^{\frac{\gamma}{\alpha}} = g_2^{s x_2} \cdot g_2^{\frac{\gamma}{\alpha}}, \\
\mathtt{K}_x' &:= g_2^{x_0} (g_2^{c_1})^{\frac{\mathtt{T}^{(i_k)}(x_1' + y_1') + x_3' + y_3' + \mathtt{I}^{(i_k)}(x_4' + y_4')}{\alpha}}
\end{aligned}$$

$$\cdot T^{\frac{\mathtt{T}^{(i_k)}y_1''+y_3''+\mathtt{I}^{(i_k)}y_4''}{\alpha}}$$
$$= g_2^{x_0+c_1\cdot\frac{\mathtt{T}^{(i_k)}(x_1'+y_1'+y_1''c_2)+x_3'+y_3'+y_3''c_2+\mathtt{I}^{(i_k)}(x_4'+y_4'+y_4''c_2)}{\alpha}}$$
$$\cdot g_2^{\frac{\gamma(\mathtt{T}^{(i_k)}y_1''+y_3''+\mathtt{I}^{(i_k)}y_4'')}{\alpha}}$$
$$= g_2^{x_0+s(\mathtt{T}^{(i_k)}x_1+x_3+\mathtt{I}^{(i_k)}x_4)}g_2^{\frac{\gamma\phi}{\alpha}}$$

$\mathtt{K}_y := T^{-1} = g_2^{-c_1c_2-\gamma} = g_2^{-sy_2-\gamma}$,

$\mathtt{K}_y' := g_2^{-y_0}(g_2^{c_1})^{-\mathtt{T}^{(i_k)}y_1'-y_3'-\mathtt{I}^{(i_k)}y_4'}$
$$\cdot T^{-\mathtt{T}^{(i_k)}y_1''-y_3''-\mathtt{I}^{(i_k)}y_4''}$$
$$= g_2^{-y_0-c_1(\mathtt{T}^{(i_k)}(y_1'+y_1''c_2)+y_3'+y_3''c_2+\mathtt{I}^{(i_k)}(y_4'+y_4''c_2))}$$
$$\cdot g_2^{-\gamma(\mathtt{T}^{(i_k)}y_1''+y_3''+\mathtt{I}^{(i_k)}y_4'')}$$
$$= g_2^{-y_0-s(\mathtt{T}^{(i_k)}y_1+y_3+\mathtt{I}^{(i_k)}y_4)}\cdot g_2^{-\gamma\phi},$$

where $T := g_2^{c_1c_2+\gamma}$, $s := c_1$, $\phi := \mathtt{T}^{(i_k)}y_1''+y_3''+\mathtt{I}^{(i_k)}y_4''$. Since y_1'', y_3'', and y_4'' are chosen uniformly at random, ϕ is uniformly distributed. □

Lemma 7 $|S_{3,q} - S_{\mathit{Final}}| \leq 2\mathsf{Adv}^{\mathsf{DDH1}}_{\mathcal{G},\mathcal{B}}(1^\lambda)$.

Proof At the beginning, a PPT adversary $\mathcal{B}$ receives an instance $(g_1, g_1^{c_1}, g_1^{c_2}, g_2, T)$ of the DDH1 problem. Then, $\mathcal{B}$ randomly chooses $y_0', x_1, y_1', x_2, y_2', x_3, y_3', x_4, y_4' \xleftarrow{\$} \mathbb{Z}_p$ and $\alpha \xleftarrow{\$} \mathbb{Z}_p^\times$, and (implicitly) sets

$x_0 := c_1,\ y_0 := x_0\alpha + y_0',\ y_1 := x_1\alpha + y_1'$,
$y_2 := x_2\alpha + y_2',\ y_3 := x_3\alpha + y_3',\ y_4 := x_4\alpha + y_4'$.

Then, $\mathcal{B}$ creates

$z := e(g_1,g_2)^{-y_0'},\ u_1 := g_1^{-y_1'},\ w_1 := g_1^{-y_2'}$,
$h_1 := g_1^{-y_3'},\ v_1 := g_1^{-y_4'}$.

$\mathcal{B}$ sends $\mathsf{prms} := (g_1, g_1^\alpha, u_1, w_1, h_1, v_1, z)$ to $\mathcal{A}$. Note that $\mathcal{B}$ does not know a part of mhk (i.e, x_0 and y_0).

When receiving a query $(\mathtt{T}, \mathtt{I})$, $\mathcal{B}$ can simulate the oracle $\mathcal{O}_{\mathrm{EXT}}$ as follows. $\mathcal{B}$ chooses $s, \phi' \xleftarrow{\$} \mathbb{Z}_p$ and $\gamma \xleftarrow{\$} \mathbb{Z}_p^\times$, and sets

$$\phi := \frac{\alpha(\phi' - x_0 - s(x_1\mathtt{T} + x_3 + x_4\mathtt{I}))}{\gamma}.$$

We then have $\phi' = x_0 + s(x_1\mathtt{T} + x_3 + x_4\mathtt{I}) + \frac{\gamma\phi}{\alpha}$. $\mathcal{B}$ computes

$\tilde{\mathtt{K}} := g_2^s$,
$\tilde{\mathtt{K}}_x := g_2^{sx_2+\frac{\gamma}{\alpha}}$,
$\tilde{\mathtt{K}}_x' := g_2^{\phi'} = g_2^{x_0+s(x_1\mathtt{T}+x_3+x_4\mathtt{I})+\frac{\gamma\phi}{\alpha}}$,
$\tilde{\mathtt{K}}_y := g_2^{-sy_2-\gamma}$,
$\tilde{\mathtt{K}}_y' := g_2^{-y_0'-s(y_1'\mathtt{T}+y_3'+y_4'\mathtt{I})-\phi'\alpha} = g_2^{-y_0-s(y_1\mathtt{T}+y_3+y_4\mathtt{I})-\gamma\phi}$.

Since ϕ' is randomly chosen, ϕ is randomly distributed from $\mathcal{A}$'s view. $\mathcal{B}$ sends $\widetilde{\mathsf{dk}}_{\mathtt{T}} := (\tilde{\mathtt{K}}, \tilde{\mathtt{K}}_x, \tilde{\mathtt{K}}_x', \tilde{\mathtt{K}}_y, \tilde{\mathtt{K}}_y')$ to $\mathcal{A}$.

Similarly, $\mathcal{B}$ can simulate the oracle $\mathcal{O}_{\mathrm{LEAK}}$ for a query $\mathtt{T} \in \mathcal{T}$ as follows. $\mathcal{B}$ chooses $s_1, s_2, \phi_1', \phi_2' \xleftarrow{\$} \mathbb{Z}_p$ and $\gamma_1, \gamma_2 \xleftarrow{\$} \mathbb{Z}_p^\times$, and sets

$$\phi_1 := \frac{\alpha(\phi_1' - x_0 - s_1(x_1\mathtt{T} + x_3))}{\gamma_1},$$
$$\phi_2 := \frac{\alpha(\phi_2' - s_2(x_1\mathtt{T} + x_3))}{\gamma_2}.$$

We then have

$$\phi_1' = x_0 + s_1(x_1\mathtt{T} + x_3) + \frac{\gamma_1\phi_1}{\alpha},$$
$$\phi_2' = s_2(x_1\mathtt{T} + x_3) + \frac{\gamma_2\phi_2}{\alpha}.$$

Then $\mathcal{B}$ computes

$\tilde{\mathtt{D}}_i := g_2^{s_i}$,
$\tilde{\mathtt{K}}_{x,i} := g_2^{s_ix_2+\frac{\gamma_i}{\alpha}}$,
$\tilde{\mathtt{D}}_{x,1}' := g_2^{\phi_1'} = g_2^{x_0+s_1(x_1\mathtt{T}+x_3)+\frac{\gamma_1\phi_1}{\alpha}}$,
$\tilde{\mathtt{D}}_{x,2}' := g_2^{\phi_2'} = g_2^{s_2(x_1\mathtt{T}+x_3)+\frac{\gamma_2\phi_2}{\alpha}}$,
$\tilde{\mathtt{D}}_{x,i}'' := g_2^{s_ix_4+\frac{\gamma_i\psi_i}{\alpha}}$,
$\tilde{\mathtt{D}}_{y,i} := g_2^{-s_iy_2-\gamma_i}$,
$\tilde{\mathtt{D}}_{y,1}' := g_2^{-y_0'-s_1(y_1'\mathtt{T}+y_3')-\phi_1'\alpha} = g_2^{-y_0-s_1(y_1\mathtt{T}+y_3)-\gamma_1\phi_1}$.
$\tilde{\mathtt{D}}_{y,2}' := g_2^{-s_2(y_1'\mathtt{T}+y_3')-\phi_2'\alpha} = g_2^{-s_2(y_1\mathtt{T}+y_3)-\gamma_2\phi_2}$,
$\tilde{\mathtt{D}}_{y,i}'' := g_2^{-s_iy_4-\gamma_i\psi_i}$.

Since ϕ_1' and ϕ_2' are randomly chosen, ϕ_1 and ϕ_2 are randomly distributed from $\mathcal{A}$'s view, respectively. $\mathcal{B}$ sends $\widetilde{\mathsf{mk}}_{\mathtt{T}} := (g_2, \{\beta_i, \beta_i'\}_{i=0}^4, \{\tilde{\mathtt{D}}_i, \tilde{\mathtt{D}}_{x,i}, \tilde{\mathtt{D}}_{x,i}', \tilde{\mathtt{D}}_{x,i}'', \tilde{\mathtt{D}}_{y,i}, \tilde{\mathtt{D}}_{y,i}', \tilde{\mathtt{D}}_{y,i}''\}_{i=1}^2)$ to $\mathcal{A}$.

$\mathcal{B}$ can also simulate the $\mathcal{O}_{\mathrm{LEAK}}$ oracle for a query $\star$ as in the proof of Lemma 4. Namely, $\mathcal{B}$ randomly chooses $\zeta_0, \zeta_0' \xleftarrow{\$} \mathbb{Z}_p$, and implicitly sets "new" noises β_0 and β_0' as follows.

$\beta_0 := \zeta_0 - x_0,\quad \beta_0' := -\zeta_0' - y_0$.

Then, $\mathcal{B}$ computes $B_{x,i} := g_2^{x_i+\beta_i}$ and $B_{y,i} := g_2^{-y_i-\beta_i'}$ with $1 \leq i \leq 4$, and sets $B_{x,0} := g_2^{\zeta_0}$ and $B_{y,0} := g_2^{\zeta_0'}$. Since $\mathcal{A}$ cannot issue any queries to the oracle $\mathcal{O}_{\mathrm{LEAK}}$ if $\mathcal{A}$ queries to the $\mathcal{O}_{\mathrm{LEAK}}$ oracle, the simulation is correct from $\mathcal{A}$'s view.

In the challenge phase, $\mathcal{B}$ receives $(M_0^*, M_1^*, \mathtt{T}^*, \mathtt{I}^*)$ from $\mathcal{A}$. $\mathcal{B}$ chooses $d \xleftarrow{\$} \{0,1\}$. $\mathcal{B}$ chooses $t, \widetilde{\mathtt{tag}}^* \xleftarrow{\$} \mathbb{Z}_p$ and computes

$\tilde{C}_M^* := M_d^* \cdot e(g_1,g_2)^{-y_0't} e(T, g_2)$,

$\tilde{C}^*_x := g_1^{\alpha t} g_1^{c_2}$, $\tilde{C}^*_y := g_1^t$,
$\tilde{C}^* := (u_1^{\mathtt{T}^*} w_1^{\widetilde{\mathtt{tag}}^*} h_1 v_1^{\mathtt{I}^*})^t (g_1^{c_2})^{x_1\mathtt{T}^*+x_2\widetilde{\mathtt{tag}}^*+x_3+x_4\mathtt{I}^*}$.

$\mathcal{B}$ sends $\widetilde{\mathsf{ct}}^*_{\mathtt{T}^*,\mathtt{I}^*} := (\tilde{C}^*_M, \tilde{C}^*_x, \tilde{C}^*_y, \tilde{C}^*, \widetilde{\mathtt{tag}}^*)$ to $\mathcal{A}$.

If $b = 0$, then the above ciphertext is semi-functional one of M^*_d by setting $\mu := c_2$. If $b = 1$, then the above ciphertext is semi-functional one of a random element of $\mathbb{G}_T$ since it holds

$$\begin{aligned}\tilde{C}^*_M &= M^*_d \cdot e(g_1,g_2)^{-y'_0 t + x_0\mu + \eta}\\ &= M^*_d \cdot e(g_1,g_2)^{x_0\alpha t - y_0 t + x_0\mu + \eta}\\ &= M^*_d \cdot e(g_1,g_2)^{x_0(\alpha t+\mu) - y_0 t} e(g_1,g_2)^{\eta}\\ &= R \cdot e(g_1,g_2)^{x_0(\alpha t+\mu) - y_0 t},\end{aligned}$$

where $R = M^*_d e(g_1,g_2)^{\eta}$.

After receiving d' from $\mathcal{A}$, $\mathcal{B}$ sends $b' = 1$ to the challenger of the DDH1 problem if $d' = d$. Otherwise, $\mathcal{B}$ sends $b' = 0$ to the challenger. □

Proof of Theorem 3. From Lemmas 3–7, we have

$$\begin{aligned}&\mathsf{Adv}^{\mathsf{ID\text{-}KI\text{-}CPA}}_{\mathcal{MIKE},\mathcal{A}}(1^\lambda)\\ &\quad\le 4\mathsf{Adv}^{\mathsf{DDH1}}_{\mathcal{G},\mathcal{B}}(1^\lambda) + (4q_T + 2q_I)\cdot \mathsf{Adv}^{\mathsf{DDH2}}_{\mathcal{G},\mathcal{B}}(1^\lambda).\end{aligned}$$ □

6.3 Proof of Theorem 4

We transition between games as in the proof of Theorem 3. Namely, we define the following games:

- $\mathsf{Game}_{\mathsf{Real}}$: This is the same as the ANO-KI-CPA game.
- Game_0: This is the same as $\mathsf{Game}_{\mathsf{Real}}$ except that the challenge ciphertext is semi-functional.
- $\mathsf{Game}_{1,k}$ $(1 \le k \le q_T)$: This is the same as $\mathsf{Game}_{1,k}$ in the proof of Theorem 3. The first k master keys are partial semi-functional, and the rest of keys are normal.
- $\mathsf{Game}_{2,k}$ $(1 \le k \le q_T)$: This is the same as $\mathsf{Game}_{2,k}$ in the proof of Theorem 3. The first k master keys are semi-functional, and the rest of keys are partial semi-functional.
- $\mathsf{Game}_{3,k}$ $(1 \le k \le q_I)$: This is the same as $\mathsf{Game}_{2,k}$ in the proof of Theorem 3. The first k decryption keys are semi-functional, and the rest of keys are partial semi-functional.
- $\mathsf{Game}_{\mathsf{Final}}$: This is the same as $\mathsf{Game}_{3,q}$ except for the following modification: $\mathcal{A}$ obtains a challenge ciphertext $\mathsf{ct}^*_{\mathtt{T}^*,\mathtt{I}^*} := (C^*_M, C^*_x, C^*_y, C^*)$ such that C^* is a random element of $\mathbb{G}_1$. This means that the challenge ciphertext does not contain any information of $\mathtt{T}^*$ and $\mathtt{I}^*$.

Let S_{Real}, $S_{1,k}$ $(0 \le k \le q_T)$, $S_{2,k}$ $(0 \le k \le q_T)$, $S_{3,k}$ $(0 \le k \le q_I)$, and S_{Final} be the probabilities that the event $b' = b$ occurs in $\mathsf{Game}_{\mathsf{Real}}$, $\mathsf{Game}_{1,k}$, $\mathsf{Game}_{2,k}$, $\mathsf{Game}_{3,k}$, and $\mathsf{Game}_{\mathsf{Final}}$, respectively. Since $|S_{\mathsf{Final}} - 1/2| = 0$, we have

$$\begin{aligned}&\mathsf{Adv}^{\mathsf{ANO\text{-}KI\text{-}CPA}}_{\mathcal{MIKE},\mathcal{A}}(1^\lambda)\\ &\le |S_{\mathsf{Real}} - S_0| + \sum_{i=1}^{q_T} |S_{1,i-1} - S_{1,i}| + \sum_{i=1}^{q_T} |S_{2,i-1} - S_{2,i}|\\ &\qquad + \sum_{i=1}^{q_I} |S_{3,i-1} - S_{3,i}| + |S_{3,q_I} - S_{\mathsf{Final}}|,\end{aligned}$$

where $S_{1,0} := S_0$, $S_{2,0} := S_{1,q_T}$, and $S_{3,0} := S_{2,q_T}$.

We can show all the transitions except for the last one as in Lemmas 3 and 6. What we have to show is the last transition.

Lemma 8 $|S_{3,q_I} - S_{\mathsf{Final}}| \le 2\mathsf{Adv}^{\mathsf{DDH1}}_{\mathcal{G},\mathcal{B}}(1^\lambda)$.

Proof At the beginning, a PPT adversary $\mathcal{B}$ receives an instance $(g_1, g_1^{c_1}, g_1^{c_2}, g_2, T)$ of the DDH1 problem. Then, $\mathcal{B}$ randomly chooses $x_0, y'_0, x_1, y'_1, x_2, y'_2, y'_3, x_4, y'_4 \xleftarrow{\$} \mathbb{Z}_p$ and $\alpha \xleftarrow{\$} \mathbb{Z}_p^\times$, and (implicitly) sets

$$\begin{aligned}&y_0 := x_0\alpha + y'_0,\ y_1 := x_1\alpha + y'_1,\ y_2 := x_2\alpha + y'_2,\\ &x_3 := c_1,\ \ y_3 := x_3\alpha + y'_3,\ y_4 := x_4\alpha + y'_4.\end{aligned}$$

Then, $\mathcal{B}$ creates

$$\begin{aligned}&z := e(g_1,g_2)^{-y'_0},\ u_1 := g_1^{-y'_1},\ w_1 := g_1^{-y'_2},\\ &h_1 := g_1^{-y'_3},\ v_1 := g_1^{-y'_4}.\end{aligned}$$

$\mathcal{B}$ sends $\mathsf{prms} := (g_1, g_1^\alpha, u_1, w_1, h_1, v_1, z)$ to $\mathcal{A}$. Note that $\mathcal{B}$ does not know a part of mhk (i.e, x_3 and y_3).

When receiving a query $(\mathtt{T}, \mathtt{I})$, $\mathcal{B}$ can simulate the oracle $\mathcal{O}_{\mathrm{EXT}}$ as follows. $\mathcal{B}$ chooses $s, \phi' \xleftarrow{\$} \mathbb{Z}_p$ and $\gamma \xleftarrow{\$} \mathbb{Z}_p^\times$, and sets

$$\phi := \frac{\alpha(\phi' - s x_3))}{\gamma} \quad (\text{and hence } \phi' = s x_3 + \frac{\gamma\phi}{\alpha}).$$

Then $\mathcal{B}$ computes

$$\begin{aligned}&\tilde{\mathtt{K}} := g_2^s,\\ &\tilde{\mathtt{K}}_x := g_2^{s x_2 + \frac{\gamma}{\alpha}},\\ &\tilde{\mathtt{K}}'_x := g_2^{x_0 + s(x_1\mathtt{T} + x_4\mathtt{I}) + \phi'} = g_2^{x_0 + s(x_1\mathtt{T} + x_3 + x_4\mathtt{I}) + \frac{\gamma\phi}{\alpha}},\\ &\tilde{\mathtt{K}}_y := g_2^{-s y_2 - \gamma},\\ &\tilde{\mathtt{K}}'_y := g_2^{-y_0 - s(y_1\mathtt{T} + y'_3 + y_4\mathtt{I}) - \phi'\alpha} = g_2^{-y_0 - s(y_1\mathtt{T} + y_3 + y_4\mathtt{I}) - \gamma\phi}.\end{aligned}$$

Since ϕ' is randomly chosen, ϕ is randomly distributed from $\mathcal{A}$'s view. $\mathcal{B}$ sends $\widetilde{\mathsf{dk}}_{\mathtt{T}} := (\tilde{\mathtt{K}}, \tilde{\mathtt{K}}_x, \tilde{\mathtt{K}}'_x, \tilde{\mathtt{K}}_y, \tilde{\mathtt{K}}'_y)$ to $\mathcal{A}$.

Similarly, $\mathcal{B}$ can simulate the oracle $\mathcal{O}_{\text{LEAK}}$ for a query $\mathtt{T}$ as follows. $\mathcal{B}$ chooses $s_1, s_2, \phi'_1, \phi'_2 \xleftarrow{\$} \mathbb{Z}_p$ and $\gamma_1, \gamma_2 \xleftarrow{\$} \mathbb{Z}_p^\times$, and sets

$$\phi_1 := \frac{\alpha(\phi'_1 - s_1 x_3)}{\gamma_1} \quad (\text{and hence } \phi'_1 = s_1 x_3 + \frac{\gamma_1 \phi_1}{\alpha}),$$
$$\phi_2 := \frac{\alpha(\phi'_2 - s_2 x_3)}{\gamma_2} \quad (\text{and hence } \phi'_2 = s_2 x_3 + \frac{\gamma_2 \phi_2}{\alpha}).$$

Then $\mathcal{B}$ computes

$$\begin{aligned}
\tilde{\mathtt{D}}_i &:= g_2^{s_i}, \\
\tilde{\mathtt{K}}_{x,i} &:= g_2^{s_i x_2 + \frac{\gamma_i}{\alpha}}, \\
\tilde{\mathtt{D}}'_{x,1} &:= g_2^{x_0 + s_1 x_1 \mathtt{T} + \phi'_1} = g_2^{x_0 + s_1(x_1 \mathtt{T} + x_3) + \frac{\gamma_1 \phi_1}{\alpha}}, \\
\tilde{\mathtt{D}}'_{x,2} &:= g_2^{s_2 x_1 \mathtt{T} + \phi'_2} = g_2^{s_2(x_1 \mathtt{T} + x_3) + \frac{\gamma_2 \phi_2}{\alpha}}, \\
\tilde{\mathtt{D}}''_{x,i} &:= g_2^{s_i x_4 + \frac{\gamma_i \psi_i}{\alpha}}, \\
\tilde{\mathtt{D}}_{y,i} &:= g_2^{-s_i y_2 - \gamma_i}, \\
\tilde{\mathtt{D}}'_{y,1} &:= g_2^{-y_0 - s_1(y_1 \mathtt{T} + y'_3) - \phi'_1 \alpha} = g_2^{-y_0 - s_1(y_1 \mathtt{T} + y_3) - \gamma_1 \phi_1}. \\
\tilde{\mathtt{D}}'_{y,2} &:= g_2^{-s_2(y_1 \mathtt{T} + y'_3) - \phi'_2 \alpha} = g_2^{-s_2(y_1 \mathtt{T} + y_3) - \gamma_2 \phi_2}, \\
\tilde{\mathtt{D}}''_{y,i} &:= g_2^{-s_i y_4 - \gamma_i \psi_i}.
\end{aligned}$$

Since ϕ'_1 and ϕ'_2 are randomly chosen, ϕ_1 and ϕ_2 are randomly distributed from $\mathcal{A}$'s view, respectively. $\mathcal{B}$ then sends $\widetilde{\mathsf{mk}}_{\mathtt{T}} := (g_2, \{\beta_i, \beta'_i\}_{i=0}^4, \{\tilde{\mathtt{D}}_i, \tilde{\mathtt{D}}_{x,i}, \tilde{\mathtt{D}}'_{x,i}, \tilde{\mathtt{D}}''_{x,i}, \tilde{\mathtt{D}}_{y,i}, \tilde{\mathtt{D}}'_{y,i}, \tilde{\mathtt{D}}''_{y,i}\}_{i=1}^2)$ to $\mathcal{A}$.

$\mathcal{B}$ can also simulate the oracle $\mathcal{O}_{\text{LEAK}}$ for a query $\star$ as in the proof of Lemma 4. Namely, $\mathcal{B}$ randomly chooses $\zeta_3, \zeta'_3 \xleftarrow{\$} \mathbb{Z}_p$, and implicitly sets "new" noises β_3 and β'_3 as follows.

$$\beta_3 := \zeta_3 - x_3, \quad \beta'_3 := -\zeta'_3 - y_3.$$

Then, $\mathcal{B}$ computes $B_{x,i} := g_2^{x_i + \beta_i}$ and $B_{y,i} := g_2^{-y_i - \beta'_i}$ with $i \in \{0, 1, 2, 4\}$, and sets $B_{x,3} := g_2^{\zeta_3}$ and $B_{y,3} := g_2^{\zeta'_3}$. Since $\mathcal{A}$ cannot issue any other queries to the oracle $\mathcal{O}_{\text{LEAK}}$ when $\mathcal{A}$ issues $\star$ to it, the simulation is correct from $\mathcal{A}$'s view.

In the challenge phase, $\mathcal{B}$ receives $(M*, \mathtt{I}_0^*, \mathtt{I}_1^*, \mathtt{T}^*)$ from $\mathcal{A}$. $\mathcal{B}$ chooses $d \xleftarrow{\$} \{0, 1\}$. $\mathcal{B}$ chooses $t, \mathtt{tag}^* \xleftarrow{\$} \mathbb{Z}_p$ and computes

$$\begin{aligned}
\tilde{C}^*_M &:= M^* \cdot e(g_1, g_2)^{(x_0 \alpha - y_0)t} e(g_1^{c_2}, g_2)^{x_0}, \\
\tilde{C}^*_x &:= g_1^{\alpha t} g_1^{c_2}, \quad \tilde{C}^*_y := g_1^t, \\
\tilde{C}^* &:= (u_1^{\mathtt{T}^*} w_1^{\mathtt{tag}^*} h_1 v_1^{\mathtt{I}^*_d})^t (g_1^{c_2})^{x_1 \mathtt{T}^* + x_2 \mathtt{tag}^* + x_4 \mathtt{I}^*_d T}.
\end{aligned}$$

$\mathcal{B}$ sends $\widetilde{\mathsf{ct}}^*_{\mathtt{T}^*, \mathtt{I}^*_d} := (\tilde{C}^*_M, \tilde{C}^*_x, \tilde{C}^*_y, \tilde{C}^*, \mathtt{tag}^*)$ to $\mathcal{A}$.

If $b = 0$, then the above ciphertext is semi-functional one of M^*_d by setting $\mu := c_2$. If $b = 1$, then $\tilde{C}^*$ is a random element of $\mathbb{G}_1$ since it holds

$$\begin{aligned}
\tilde{C}^* &= (u_1^{\mathtt{T}^*} w_1^{\mathtt{tag}^*} h_1 v_1^{\mathtt{I}^*_d})^t (g_1^{c_2})^{x_1 \mathtt{T}^* + x_2 \mathtt{tag}^* + x_4 \mathtt{I}^*_d T} \\
&= (u_1^{\mathtt{T}^*} w_1^{\mathtt{tag}^*} h_1 v_1^{\mathtt{I}^*_d})^t (g_1^{c_2})^{x_1 \mathtt{T}^* + x_2 \mathtt{tag}^* + x_3 + x_4 \mathtt{I}^*_d} g_1^{\eta}.
\end{aligned}$$

This means any information of $\mathtt{I}^*_d$ and $\mathtt{T}^*$ underlying $\widetilde{ct}^*_{\mathtt{T}^*, \mathtt{I}^*_d}$ is completely hidden from $\mathcal{A}$'s view.

After receiving d' from $\mathcal{A}$, $\mathcal{B}$ sends $b' = 1$ to the challenger of the DDH1 problem if $d' = d$. Otherwise, $\mathcal{B}$ sends $b' = 0$ to the challenger. □

Proof of Theorem 4. From Lemmas 3–6 and 8, we have

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{ANO\text{-}KI\text{-}CPA}}_{\mathcal{MIKE}, \mathcal{A}}(1^\lambda) \\
&\quad \leq 4\mathsf{Adv}^{\mathsf{DDH1}}_{\mathcal{G}, \mathcal{B}}(1^\lambda) + (4q_T + 2q_I) \cdot \mathsf{Adv}^{\mathsf{DDH2}}_{\mathcal{G}, \mathcal{B}}(1^\lambda).
\end{aligned}$$ □

7 Conclusion

In this paper, we first proposed key-updatable public-key encryption with keyword search as one solution against key leakage problem for IoT environments. Specifically we proposed two model: the key-evolution model and the key insulation model. In each model, we defined the model and security notions, and showed the generic construction. The former model is conceptually simple. On the one hand, the latter model is preferable for practical use since redistribution of public key is not needed. We show implementation results on Raspberry Pi of our proposals. Our instantiation in the key evolution model is fastest in the sense of running time. On the other hand, the execution time of each core algorithm of our instantiation in the key insulation model (on Raspberry Pi) is less than 550 msec except for the setup and key update algorithms. Hence, we conclude that all the instantiations could be applicable for IoT devices.

Acknowledgements We would like to thank the anonymous reviewers for useful comments. The first, second, and third authors were supported by Grant-in-Aid for Scientific Research (C) Grant Number JP17K00189. The last author was supported by JSPS Research Fellowship for Young Scientists, Grant-in-Aid for JSPS Fellows Grant Number JP16J10532, and Grant-in-Aid for Young Scientists (B) Grant Number JP17K12697.

Compliance with Ethical Standards The author Hiroaki Anada declares that he has no conflict of interest. The author Akira Kanaoka declares that he has no conflict of interest. The author Natsume Matsuzaki declares that she has no conflict of interest. The author Yohei Watanabe declares that he has no conflict of interest.

Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., Shi, H.: Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In: Shoup, V. (ed.) Advances in Cryptology – CRYPTO 2005. vol. 3621, pp. 205–222. Springer (2005)
2. Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., Shi, H.: Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. J. Cryptology 21(3), 350–391 (2008)
3. Anada, H., Kanaoka, A., Matsuzaki, N., Watanabe, Y.: Key-updatable public-key encryption with keyword search: Models and generic constructions. In: Susilo, W., Yang, G. (eds.) Information Security and Privacy. pp. 341–359. Springer International Publishing, Cham (2018)
4. Baek, J., Safavi-Naini, R., Susilo, W.: Public key encryption with keyword search revisited. In: ICCSA 2008, Part I. pp. 1249–1259 (2008)
5. Blaze, M., Bleumer, G., Strauss, M.: Divertible protocols and atomic proxy cryptography. In: Nyberg, K. (ed.) Advances in Cryptology – EUROCRYPT'98. vol. 1403, pp. 127–144. Springer Berlin Heidelberg (1998)
6. Boldyreva, A., Goyal, V., Kumar, V.: Identity-based encryption with efficient revocation. In: Proc. of CCS'08. pp. 417–426. ACM, New York, NY, USA (2008)
7. Boneh, D., Crescenzo, G.D., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Advances in Cryptology - EUROCRYPT 2004. pp. 506–522 (2004)
8. Boneh, D., Franklin, M.K.: Identity-based encryption from the weil pairing. In: Advances in Cryptology – CRYPTO'01. pp. 213–229. Springer-Verlag (2001)
9. Boneh, D., Kushilevitz, E., Ostrovsky, R., III, W.E.S.: Public key encryption that allows PIR queries. In: Advances in Cryptology - CRYPTO 2007. pp. 50–67 (2007)
10. Byun, J.W., Rhee, H.S., Park, H.A., Lee, D.H.: Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In: Jonker, W., Petković, M. (eds.) Secure Data Management. pp. 75–83. Springer Berlin Heidelberg (2006)
11. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In: Biham, E. (ed.) Advances in Cryptology – EUROCRYPT 2003. LNCS, vol. 2656, pp. 255–271. Springer Berlin Heidelberg (2003)
12. Cheon, J.H., Hopper, N., Kim, Y., Osipkov, I.: Provably secure timed-release public key encryption. ACM Trans. Inf. Syst. Secur. 11(2), 4:1–4:44 (May 2008)
13. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: FOCS'95. pp. 41–50 (1995)
14. Dodis, Y., Franklin, M., Katz, J., Miyaji, A., Yung, M.: A generic construction for intrusion-resilient public-key encryption. In: Okamoto, T. (ed.) Topics in Cryptology – CT-RSA 2004. vol. 2964, pp. 81–98. Springer Berlin Heidelberg (2004)
15. Dodis, Y., Katz, J., Xu, S., Yung, M.: Key-insulated public key cryptosystems. In: Knudsen, L. (ed.) Advances in Cryptology – EUROCRYPT 2002. vol. 2332, pp. 65–82. Springer Berlin Heidelberg (2002)
16. Dodis, Y., Katz, J., Xu, S., Yung, M.: Strong key-insulated signature schemes. In: Desmedt, Y. (ed.) PKC 2003. vol. 2567, pp. 130–144. Springer (2003)
17. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G., Chaum, D. (eds.) Advances in Cryptology – CRYPTO'84. vol. 196, pp. 10–18. Springer Berlin Heidelberg (1985)
18. Emura, K., Phong, L.T., Watanabe, Y.: Keyword revocable searchable encryption with trapdoor exposure resistance and re-generateability. In: 2015 IEEE Trustcom/BigDataSE/ISPA. vol. 1, pp. 167–174 (Aug 2015)
19. Green, M., Ateniese, G.: Identity-based proxy re-encryption. In: ACNS 2007. pp. 288–306 (2007)
20. Hanaoka, Y., Hanaoka, G., Shikata, J., Imai, H.: Identity-based hierarchical strongly key-insulated encryption and its application. In: Roy, B. (ed.) Advances in Cryptology – ASIACRYPT 2005. vol. 3788, pp. 495–514. Springer (2005)
21. Jutla, C.S., Roy, A.: Shorter quasi-adaptive NIZK proofs for linear subspaces. In: Sako, K., Sarkar, P. (eds.) Advances in Cryptology – ASIACRYPT 2013. LNCS, vol. 8269, pp. 1–20. Springer Berlin Heidelberg (2013)
22. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Advances in Cryptology - CRYPTO '96. pp. 104–113 (1996)
23. Lewko, A.B.: Tools for simulating features of composite order bilinear groups in the prime order setting. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. vol. 7237, pp. 318–335. Springer (2012)
24. Libert, B., Vergnaud, D.: Adaptive-id secure revocable identity-based encryption. In: Fischlin, M. (ed.) Topics in Cryptology – CT-RSA 2009. vol. 5473, pp. 1–15. Springer Berlin Heidelberg (2009)
25. Naor, D., Naor, M., Lotspiech, J.: Revocation and tracing schemes for stateless receivers. In: Kilian, J. (ed.) Advances in Cryptology – CRYPTO 2001. vol. 2139, pp. 41–62. Springer Berlin Heidelberg (2001)
26. National Institute of Standards and Technology: Nist special publication 800-57 part 1, revision 4, recommendation for key management part 1: General (2013),
27. Seo, J.H., Emura, K.: Revocable identity-based encryption revisited: Security model and construction. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. vol. 7778, pp. 216–234. Springer Berlin Heidelberg (2013)
28. Shikata, J., Watanabe, Y.: Identity-based encryption with hierarchical key-insulation in the standard model. Designs, Codes and Cryptography (Jun 2018), to appear. First online: 8th June 2018.
29. Tang, Q.: Towards forward security properties for peks and ibe. In: Foo, E., Stebila, D. (eds.) ACISP 2015. vol. 9144, pp. 127–144. Springer (2015)
30. Watanabe, Y., Shikata, J.: Identity-based hierarchical key-insulated encryption without random oracles. In: Cheng, C.M., Chung, K.M., Persiano, G., Yang, B.Y. (eds.) PKC 2016, Part I. LNCS, vol. 9614, pp. 255–279. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
31. Waters, B.: Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In: Halevi, S. (ed.) Advances in Cryptology – CRYPTO 2009. vol. 5677, pp. 619–636. Springer Berlin Heidelberg (2009)

A Proof of Lemma 1

We construct a PPT adversary $\mathcal{B}$ which breaks the IND-CPA security of $\mathcal{PKE}$ using a PPT adversary $\mathcal{A}$ which wins G_2 or G_3.

Setup. $\mathcal{B}$ guesses i^* such that i^* is a time period when computing the challenge ciphertext, and the guess is correct since Fail does not occur. Without loss of generality, we here assume $i^* \neq 1$. When receiving $(\mathsf{par}_{\mathrm{PKE}}, \mathsf{ek}^*)$, $\mathcal{B}$ computes $(\mathsf{ek}_1, \mathsf{dk}_1) \leftarrow \mathsf{G}(\mathsf{par}_{\mathrm{PKE}})$, $\mathsf{par}_{\mathrm{PEKS}} \leftarrow \mathsf{Setup}_{\mathrm{PEKS}}(1^\lambda)$, and $(\mathsf{mpk}_1, \mathsf{msk}_1) \leftarrow \mathsf{KeyGen}_{\mathrm{PEKS}}(\mathsf{par}_{\mathrm{PEKS}})$, and chooses $\mathsf{H} \xleftarrow{\$} \mathcal{H}$, $\mathcal{A}$ then sends $\mathsf{pk}_1 := (\mathsf{par}_{\mathrm{PKE}}, \mathsf{par}_{\mathrm{PEKS}}, \mathsf{H}, \mathsf{ek}_1, \mathsf{mpk}_1)$ to $\mathcal{A}$. $\mathcal{B}$ stores $\mathsf{sk}_1 := (\mathsf{dk}_1, \mathsf{msk}_1)$.

Oracle simulation. $\mathcal{B}$ simulates each oracle as follows.

- $\mathcal{O}_{\mathrm{KG}}$: If $\mathsf{ctr} \in \{1, \ldots, i^*-2\}$, $\mathcal{B}$ computes $(\mathsf{ek}_{\mathsf{ctr}+1}, \mathsf{dk}_{\mathsf{ctr}+1}) \leftarrow \mathsf{G}(\mathsf{par}_{\mathrm{PKE}})$ and $(\mathsf{mpk}_{\mathsf{ctr}+1}, \mathsf{msk}_{\mathsf{ctr}+1}) \leftarrow \mathsf{KeyGen}_{\mathrm{PEKS}}(\mathsf{par}_{\mathrm{PEKS}})$, and returns $\mathsf{pk}_{\mathsf{ctr}+1} := (\mathsf{par}_{\mathrm{PKE}}, \mathsf{par}_{\mathrm{PEKS}}, \mathsf{H}, \mathsf{ek}_{\mathsf{ctr}+1}, \mathsf{mpk}_{\mathsf{ctr}+1})$ and $\mathsf{rk}_{\mathsf{ctr}\to\mathsf{ctr}+1} := \mathsf{dk}_{\mathsf{ctr}}$ to $\mathcal{A}$. It stores $\mathsf{sk}_{\mathsf{ctr}+1} := (\mathsf{dk}_{\mathsf{ctr}+1}, \mathsf{msk}_{\mathsf{ctr}+1})$, and sets $\mathsf{ctr} := \mathsf{ctr}+1$. If $\mathsf{ctr} = i^*-1$, $\mathcal{B}$ computes $(\mathsf{mpk}_{i^*}, \mathsf{msk}_{i^*}) \leftarrow \mathsf{KeyGen}_{\mathrm{PEKS}}(\mathsf{par}_{\mathrm{PEKS}})$, and returns $\mathsf{pk}_{i^*} := (\mathsf{par}_{\mathrm{PKE}}, \mathsf{par}_{\mathrm{PEKS}}, \mathsf{H}, \mathsf{ek}^*, \mathsf{mpk}_{i^*})$ and $\mathsf{rk}_{i^*-1\to i^*} := \mathsf{dk}_{i^*-1}$ to $\mathcal{A}$. $\mathcal{B}$ stores only msk_{i^*}, and sets $\mathsf{ctr} := i^*$. Note that $\mathcal{B}$ does not know dk_{i^*}.
- $\mathcal{O}_{\mathrm{KL}}$: For a query $j \in \{1, \ldots, \mathsf{ctr}-1\}$, $\mathcal{B}$ returns sk_j.
- $\mathcal{O}_{\mathrm{TD}}$: For $(w, j) \in \mathcal{W} \times \{1, \ldots, \mathsf{ctr}\}$, $\mathcal{B}$ returns $\mathsf{Trapdoor}_{\mathrm{PEKS}}(\mathsf{msk}_j, \mathsf{H}(w))$.

Challenge. $\mathcal{B}$ receives (w_0^*, w_1^*) from $\mathcal{A}$ and randomly chooses $\beta \leftarrow \{0,1\}$. $\mathcal{B}$ chooses a zero-bit string $0^{\log|\mathcal{Y}|}$ whose length is the same as the output of H (we assume $0^{\log|\mathcal{Y}|}$ can be efficiently encoded into an element of $\mathcal{Y}$). $\mathcal{B}$ sends $(\hat{w}_0^*, \hat{w}_1^*) := (\mathsf{H}(w_\beta^*), 0^{\log|\mathcal{Y}|})$ to the challenger of $\mathcal{PKE}$ as challenge plaintexts. The challenger chooses $b \xleftarrow{\$} \{0,1\}$, and returns $\mathsf{ct}_{\mathsf{ctr}} \leftarrow \mathsf{E}(\mathsf{ek}^*, \hat{w}_b^*)$ to $\mathcal{B}$. $\mathcal{B}$ computes $\mathsf{ct}_{w_\beta^*,\mathsf{ctr}} \leftarrow \mathsf{Enc}_{\mathrm{PEKS}}(\mathsf{mpk}_{\mathsf{ctr}}, \mathsf{H}(w_\beta^*))$, and returns $\mathsf{c}^{(0)}_{w_\beta^*,\mathsf{ctr}} := (\mathsf{ct}_{\mathsf{ctr}}, \mathsf{ct}_{w_\beta^*,\mathsf{ctr}})$ to $\mathcal{A}$.

Output. If $\mathcal{A}$'s output β' satisfies $\beta' = \beta$, $\mathcal{B}$ outputs $b' = 0$. Otherwise, $\mathcal{B}$ outputs $b' = 1$.

If $b = 0$, $\mathsf{c}^{(0)}_{w_\beta^*,i^*}$ is the challenge ciphertext in G_2 where Fail does not occur. On the other hand, if $b = 1$, $\mathsf{c}^{(0)}_{w_\beta^*,i^*}$ is the challenge ciphertext in G_3 where Fail does not occur Therefore, we have

$$
\begin{aligned}
&\mathsf{Adv}^{\mathsf{CPA}}_{\mathcal{PKE},\mathcal{B}}(1^\lambda) \\
&= \left|\Pr[\mathsf{Exp}^{\mathsf{CPA}}_{\mathcal{PKE},\mathcal{B}}(1^\lambda) = 1] - \frac{1}{2}\right| \\
&= \left|\Pr[b' = b] - \frac{1}{2}\right| \\
&= \left|\Pr[b' = 0 \wedge b = 0] + \Pr[b' = 1 \wedge b = 1] - \frac{1}{2}\right| \\
&= \left|\frac{1}{2}\Pr[b' = 0 \mid b = 0] + \frac{1}{2}\Pr[b' = 1 \mid b = 1] - \frac{1}{2}\right| \\
&= \left|\frac{1}{2}\Pr[b' = 0 \mid b = 0] + \frac{1}{2}\left(1 - \Pr[b' = 0 \mid b = 1]\right) - \frac{1}{2}\right| \\
&= \frac{1}{2}\left|\Pr[b' = 0 \mid b = 0] - \Pr[b' = 0 \mid b = 1]\right| \\
&= \frac{1}{2}\left|\Pr[\beta' = \beta \mid b = 0] - \Pr[\beta' = \beta \mid b = 1]\right| \\
&= \frac{1}{2}\left|\Pr[\mathsf{S}_2 \mid \neg\mathsf{Fail}] - \Pr[\mathsf{S}_3 \mid \neg\mathsf{Fail}]\right|.
\end{aligned}
$$

Hence, we have

$$\left|\Pr[\mathsf{S}_2 \mid \neg\mathsf{Fail}] - \Pr[\mathsf{S}_3 \mid \neg\mathsf{Fail}]\right| = 2\mathsf{Adv}^{\mathsf{CPA}}_{\mathcal{PKE},\mathcal{B}}(1^\lambda). \qquad \square$$

B Proof of Lemma 2

We construct a PPT adversary $\mathcal{B}$ which breaks the IND-CKA security of $\mathcal{PEKS}$ using a PPT adversary $\mathcal{A}$ which wins G_3 when Fail does not occur.

Setup. This procedure is almost the same as that in the proof of Lemma 1. $\mathcal{B}$ guesses i^* such that i^* is a time period when generating the challenge ciphertext, and the guess is correct since Fail does not occur. Without loss of generality, we here assume $i^* \neq 1$. When receiving $(\mathsf{par}_{\mathrm{PEKS}}, \mathsf{mpk}^*)$, $\mathcal{B}$ runs $\mathsf{par}_{\mathrm{PKE}} \leftarrow \mathsf{PG}(1^\lambda)$, $(\mathsf{ek}_1, \mathsf{dk}_1) \leftarrow \mathsf{G}(\mathsf{par}_{\mathrm{PKE}})$, and $(\mathsf{mpk}_1, \mathsf{msk}_1) \leftarrow \mathsf{KeyGen}_{\mathrm{PEKS}}(\mathsf{par}_{\mathrm{PEKS}})$, and chooses $\mathsf{H} \xleftarrow{\$} \mathcal{H}$. $\mathcal{B}$ sends $\mathsf{pk}_1 := (\mathsf{par}_{\mathrm{PKE}}, \mathsf{par}_{\mathrm{PEKS}}, \mathsf{H}, \mathsf{ek}_1, \mathsf{mpk}_1)$ to $\mathcal{A}$, and stores $\mathsf{sk}_1 := (\mathsf{dk}_1, \mathsf{msk}_1)$.

Oracle simulation $\mathcal{B}$ simulates each oracle as follows.

- $\mathcal{O}_{\mathrm{KG}}$: If $\mathsf{ctr} \in \{1, \ldots, i^*-2\}$, $\mathcal{B}$ computes $(\mathsf{ek}_{\mathsf{ctr}+1}, \mathsf{dk}_{\mathsf{ctr}+1}) \leftarrow \mathsf{G}(\mathsf{par}_{\mathrm{PKE}})$ and $(\mathsf{mpk}_{\mathsf{ctr}+1}, \mathsf{msk}_{\mathsf{ctr}+1}) \leftarrow \mathsf{KeyGen}_{\mathrm{PEKS}}(\mathsf{par}_{\mathrm{PEKS}})$, and returns $\mathsf{pk}_{\mathsf{ctr}+1} := (\mathsf{par}_{\mathrm{PKE}}, \mathsf{par}_{\mathrm{PEKS}}, \mathsf{H}, \mathsf{ek}_{\mathsf{ctr}+1}, \mathsf{mpk}_{\mathsf{ctr}+1})$ and $\mathsf{rk}_{\mathsf{ctr}\to\mathsf{ctr}+1} := \mathsf{dk}_{\mathsf{ctr}}$ to $\mathcal{A}$. It stores $\mathsf{sk}_{\mathsf{ctr}+1} := (\mathsf{dk}_{\mathsf{ctr}+1}, \mathsf{msk}_{\mathsf{ctr}+1})$, and sets $\mathsf{ctr} := \mathsf{ctr}+1$. If $\mathsf{ctr} = i^*-1$, $\mathcal{B}$ computes $(\mathsf{ek}_{i^*}, \mathsf{dk}_{i^*}) \leftarrow \mathsf{G}(\mathsf{par}_{\mathrm{PKE}})$, and returns $\mathsf{pk}_{i^*} := (\mathsf{par}_{\mathrm{PKE}}, \mathsf{par}_{\mathrm{PEKS}}, \mathsf{H}, \mathsf{ek}_{i^*}, \mathsf{mpk}^*)$ and $\mathsf{rk}_{i^*-1\to i^*} := \mathsf{dk}_{i^*-1}$ to $\mathcal{A}$. $\mathcal{B}$ stores only dk_{i^*}, and sets $\mathsf{ctr} := i^*$. Note that $\mathcal{B}$ does not know msk_{i^*}.
- $\mathcal{O}_{\mathrm{KL}}$: For a query $j \in \{1, \ldots, \mathsf{ctr}-1\}$, $\mathcal{B}$ returns sk_j.
- $\mathcal{O}_{\mathrm{TD}}$: If $\mathsf{ctr} \neq i^*$, for a query $(w, j) \in \mathcal{W} \times \{1, \ldots, \mathsf{ctr}\}$, $\mathcal{B}$ returns $\mathsf{Trapdoor}_{\mathrm{PEKS}}(\mathsf{msk}_j, \mathsf{H}(w))$. If $\mathsf{ctr} = i^*$, $\mathcal{B}$ simulates the oracle as follows. For a query $(w, j) \in \mathcal{W} \times \{1, \ldots, \mathsf{ctr}\}$, if $j \neq i^*$, $\mathcal{B}$ returns $\mathsf{Trapdoor}_{\mathrm{PEKS}}(\mathsf{msk}_j, \mathsf{H}(w))$. Otherwise, $\mathcal{B}$ sends w to $\mathcal{O}_{\mathrm{TD}}$ of $\mathcal{PEKS}$ to get $\mathsf{t}^*_w \leftarrow \mathsf{Trapdoor}_{\mathrm{PEKS}}(\mathsf{msk}^*, \mathsf{H}(w))$, and transfers it to $\mathcal{A}$.

Challenge. When receiving (w_0^*, w_1^*) from $\mathcal{A}$, $\mathcal{B}$ sends $(\mathsf{H}(w_0^*), \mathsf{H}(w_1^*))$ to the challenger of $\mathcal{PEKS}$. The challenger randomly chooses $b \xleftarrow{\$} \{0,1\}$, and returns $\mathsf{ct}_{w_b^*,\mathsf{ctr}} \leftarrow \mathsf{Enc}_{\mathrm{PEKS}}(\mathsf{mpk}^*, \mathsf{H}(w_b^*))$ to $\mathcal{B}$. $\mathcal{B}$ computes $\mathsf{ct}_{\mathsf{ctr}} \leftarrow \mathsf{E}(\mathsf{ek}_{\mathsf{ctr}}, 0^{\log|\mathcal{Y}|})$, and returns $\mathsf{c}^{(0)}_{w_\beta^*,\mathsf{ctr}} := (\mathsf{ct}_{\mathsf{ctr}}, \mathsf{ct}_{w_\beta^*,\mathsf{ctr}})$ to $\mathcal{A}$.

Output $\mathcal{B}$ outputs b', the output of $\mathcal{A}$ as is.

The success probability of $\mathcal{B}$ for the IND-CKA game is the same as that of $\mathcal{A}$ for G_3. Therefore, we have

$$\left|\Pr[\mathsf{S}_3 \mid \neg\mathsf{Fail}] - \frac{1}{2}\right| = \mathsf{Adv}^{\mathsf{CKA}}_{\mathcal{PEKS},\mathcal{B}}(1^\lambda). \qquad \square$$

C DBDH Assumption

The decisional bilinear Diffie-Hellman (DBDH) assumption is defined as follows. Let $\mathcal{A}$ be a PPT adversary, and we consider the following game against $\mathcal{A}$.

$\mathsf{Exp}^{\mathsf{DBDH}}_{\mathcal{G},\mathcal{A}}(1^\lambda)$

$D := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow \mathcal{G}(1^\lambda)$

$c_1, c_2, c_3, \mu \xleftarrow{\$} \mathbb{Z}_p, \quad b \xleftarrow{\$} \{0,1\}$

if $b = 0$ **then** $T := e(g_1, g_2)^{c_1 c_2 c_3}$

else $T := e(g_1, g_2)^{\mu}$

$b' \leftarrow \mathcal{A}(D, g_1^{c_1}, g_1^{c_2}, g_1^{c_3}, g_2^{c_1}, g_2^{c_2}, g_2^{c_3}, T)$

if $b' = b$ **return** 1 **else return** 0

Definition 14 (DBDH assumption) *We say that the* DBDH *assumption relative to a generator $\mathcal{G}$ holds if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{DBDH}}_{\mathcal{G},\mathcal{A}}(1^\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{DBDH}}_{\mathcal{G},\mathcal{A}}(1^\lambda) = 1] - 1/2|$ is negligible in λ.*